

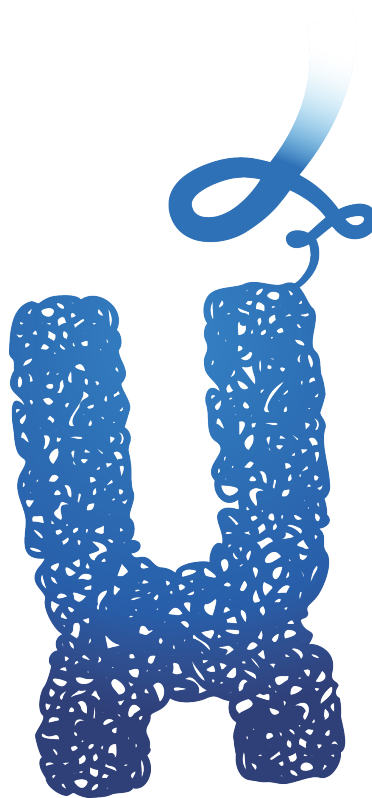
Analyzing Epigenomic Data in a Large-Scale Context

Dissertation

zur Erlangung des Grades
des Doktors der Naturwissenschaften (Dr. rer. nat.)
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von

Felipe Fernandes Albrecht



Saarbrücken, Dezember 2019

Tag des Kolloquiums: 09.12.2019

Dekan: Prof. Dr. Sebastian Hack

Prüfungsausschuss:

Vorsitzender: Prof. Dr. Olga Kalinina

Berichterstatter: Prof. Dr. Dr. Thomas Lengauer
Prof. Dr. Volkhard Helms
Prof. Dr. Benedikt Brors

Akademischer Mitarbeiter: Dr. Peter Ebert

兀然無事坐
春夾草自生

法網

「こつねんとして無事に坐すれば、春来たつて草おのずから生ず」と読み、「一切の計らいを捨て、じっと座禅をしていれば、春になって草が自然に萌えいであるように、いつか必ず悟りをえることができるでしょう。」という意味です。

"Sitting quietly, doing nothing, the spring comes, the grass grows by itself."

Abstract

While large amounts of epigenomic data are publicly available, their retrieval in a form suitable for downstream analysis is a bottleneck in current research. In a typical analysis, users are required to download huge files that span the entire genome, even if they are only interested in a small subset (e.g., promoter regions) or an aggregation thereof. Moreover, complex operations on genome-level data are not always feasible on a local computer due to resource limitations.

The *DeepBlue Epigenomic Data Server* mitigates this issue by providing a robust server that affords a powerful *API* for searching, filtering, transforming, aggregating, enriching, and downloading data from several epigenomic consortia. Furthermore, its main component implements scalable data storage and Manipulation methods that scale with the increasing amount of epigenetic data, thereby making it the ideal resource for researchers that seek to integrate epigenomic data into their analysis workflow.

This work also presents companion tools that utilize the *DeepBlue API* to enable users not proficient in scripting or programming languages to analyze epigenomic data in a user-friendly way: (i) an *R/Bioconductor* package that integrates *DeepBlue* into the *R* analysis workflow. The extracted data are automatically converted into suitable *R* data structures for downstream analysis and visualization within the *Bioconductor* framework; (ii) a web portal that enables users to search, select, filter and download the epigenomic data available in the *DeepBlue Server*. This interface provides elements, such as data tables, grids, data selections, developed for empowering users to find the required epigenomic data in a straightforward interface; (iii) *DIVE*, a web data analysis tool that allows researchers to perform large-epigenomic data analysis in a programming-free environment. *DIVE* enables users to compare their datasets to the datasets available in the *DeepBlue Server* in an intuitive interface, which summarizes the comparison of hundreds of datasets in a simple chart. Furthermore, these tools are integrated, being capable of sharing results among themselves, creating a powerful large-scale epigenomic data analysis environment.

The *DeepBlue Epigenomic Data Server* and its ecosystem was well received by the International Human Epigenome Consortium and already attracted much attention by the epigenomic research community with currently 160 registered users and more than three million anonymous workflow processing requests since its release.

Während große Mengen epigenomischer Daten öffentlich verfügbar sind, ist ihre Abfrage in einer für die Downstream-Analyse geeigneten Form ein Engpass in der aktuellen Forschung. Bei einer typischen Analyse müssen Benutzer riesige Dateien herunterladen, die das gesamte Genom umfassen, selbst wenn sie nur an einer kleinen Teilmenge (z.B., Promotorregionen) oder einer Aggregation davon interessiert sind. Darüber hinaus sind komplexe Vorgänge mit Daten auf Genomebene aufgrund von Ressourceneinschränkungen auf einem lokalen Computer nicht immer möglich.

Der *DeepBlue Epigenomic Data Server* behebt dieses Problem, indem er eine leistungsstarke API zum Suchen, Filtern, Umwandeln, Aggregieren, Anreichern und Herunterladen von Daten verschiedener epigenomischer Konsortien bietet. Darüber hinaus implementiert der *DeepBlue-Server* skalierbare Datenspeicherungs- und manipulationsmethoden, die der zunehmenden Menge epigenetischer Daten gerecht werden. Dadurch ist der DeepBlue Server ideal für Forscher geeignet, die die aktuellen epigenomischen Ressourcen in ihren Analyse-Workflow integrieren möchten.

In dieser Arbeit werden zusätzlich Begleittools vorgestellt, die die *DeepBlue-API* verwenden, um Benutzern, die sich mit Scripting oder Programmiersprachen nicht auskennen, die Möglichkeit zu geben, epigenomische Daten auf benutzerfreundliche Weise zu analysieren: (i) ein R/Bioconductor-Paket, das *DeepBlue* in den R-Analyse-Workflow integriert. Die extrahierten Daten werden automatisch in geeignete R-Datenstrukturen für die Downstream-Analyse und Visualisierung innerhalb des Bioconductor-Frameworks konvertiert; (ii) ein Webportal, über das Benutzer die auf dem *DeepBlue Server* verfügbaren epigenomischen Daten suchen, auswählen, filtern und herunterladen können. Diese Schnittstelle bietet Elemente wie Datentabellen, Raster, Datenselektionen, mit denen Benutzer die erforderlichen epigenomischen Daten in einer einfachen Schnittstelle finden können; (iii) *DIVE*, ein Webdatenanalysetool, mit dem Forscher umfangreiche epigenomische Datenanalysen in einer programmierungsfreien Umgebung durchführen können. Mit *DIVE* können Benutzer ihre Datensätze mit den im *DeepBlue Server* verfügbaren Datensätzen in einer intuitiven Benutzeroberfläche vergleichen. Dabei kann der Vergleich hunderter Datensätze in einem Diagramm ausgedrückt werden. Aufgrund der großen Datenmenge, die in *DIVE* verfügbar ist, werden Methoden bereitgestellt, mit denen die ähnlichsten Datensätze für eine vergleichende Analyse vorgeschlagen werden können. Alle zuvor genannten Tools sind miteinander integriert, so dass sie die Ergebnisse untereinander austauschen können, wodurch eine leistungsstarke Umgebung für die Analyse epigenomischer Daten entsteht.

Der *DeepBlue Epigenomic Data Server* und sein Ökosystem wurden vom International Human Epigenome Consortium äußerst gut aufgenommen und erreichten seit ihrer Veröffentlichung große Aufmerksamkeit bei der epigenomischen Forschungsgemeinschaft mit derzeit 160 registrierten Benutzern und mehr als drei Millionen anonymen Verarbeitungsanforderungen.

Acknowledgments

There are many people I want to thank. Each one was responsible for supporting me during this path in a different way but all in a meaningful form. For facilitating the task of thanking all them, I will try to follow a chronological order.

Firstly I wish to thank my father who taught me how to use and love numbers and my mother who taught me how to use and love words. Both together with my dear sister Lara always giving me all support and love. Sorrowfully my father is not among us anymore, but his values and character live in me, and I wish that he could see this work

I wish to express my gratitude and admiration to Professor Thomas Lengauer, who always trusted me, allowing me to do my work and extracting the best of me. I could not be more grateful for having this great mentor in my life.

I wish to thank Konstantin Halachev for mentoring me at the beginning of my Ph.D., Christoph Bock for guiding me through many questions on computational epigenetics, Peter Ebert for sharing the same office for more than six years and for having endless discussions, and Markus List for being my collaboration partner in many projects and guiding me during the writing of this thesis.

I am grateful for having so many great colleagues in the computational epigenetics group: Peter, Markus, Konstantin, Fabian, Karl, Pavlo, Yassen, Michael, Lars, Marcel, and Christoph. I cannot forget my dear colleagues from the Computational Biology and Applied Algorithmics Department at MPII: Olga Voitenko, Nora, Prabhav, Tomas, Bastian, Matthias, Lisa, Sarvesh, Alejandro, Ana, Olga Kalina, and Nico. Each of you is part of my happy memories at MPII.

I appreciate all support from Ruth Schnepfen-Christmann in all administrative issues that I had, the excellent technical help from Joachim Büch and Georg Friedric, the assistance with apartment issues gave by Roxane Wetzels, and the many moments that Krista Ames dedicated for improving my English.

I wish to thank the Professors from the University of Saarland Jörn Walter, for helping me to connect the bioinformatics to the wet labs, and Volkhard Helms for accepting to review this thesis. I want to thank Michelle Carnell for all support during my years at the University of Saarland, and Suzanne and Stefanie for the excellent assistance for submitting and defending this thesis.

I wish to thank Professor Benedikt Brors for accepting to review this thesis as well for together with Professor Chris Lawrenz support the installation of DeepBlue in the deNBI environment. I want to include my thanks to Professor Guillaume Bourque, David Bujold from the McGill for their excellent job in the IHEC projects and for sharing many exciting discussions.

From deep of my heart, I want to thank Professor Mikita Suyama from the Kyushu University for the wonderful months that I stayed in his lab. His knowledge, kindness, and care about all his students is a source of inspiration. I also wish to thank all colleagues that I had there for the hospitality and happy memories, specially Kikutake for explaining me so many things about baseball and Japanese culture, Kim for teaching me about Korean culture and pokemon, and Yuki for becoming a friend.

I am grateful to my wife, Rie, for her patience during the weekends where we had to stay at home for me to work on this thesis, for her tolerance of marrying a "Ph.D. Student" who is always worried about his thesis, for giving me the stability that I need, and above all, for her love.

I want to thank all my colleagues from Roche, for their patience of listening to me talking about my thesis, for trusting me that I could finish it, and mainly for all the support during this last year. Specially, I want to thank Miro for his understanding and support.

Finally, a special thank to the following people that revised this work, giving many constructive suggestions: Markus List, Lara Schneider, Nora Speicher, Miroslav Nikolov, Peter Ebert, Tomas Bastys, Honsho Masanori, Shinya Oki, Venkateswaran Sri-ram, Fabiola Rodriguez Calvino, Alexandre Strube, Daniel Tralamazza. I wish to thank Fabian Müller who supported this work with this gorgeous L^AT_EXtemplate and happily shared his figures.

Large parts of this work have been conducted within the context of the DEEP (BMBF Grant 01KU1216A) and BLUEPRINT (EU Grant HEALTH-F5-2011-282510) projects. Financial support has also been provided by basic funding of the Max Planck Society. During my research visit at Professor Mikita Suyama's lab, I was supported by the Friendship Scholarship by the Kyushu University.

Contents

1	Introduction	1
1.1	Thesis goals and outline	2
2	Computational Epigenetics	3
2.1	Epigenetics	3
2.1.1	Epigenetic Elements	4
2.2	Epigenomic Data	7
2.2.1	Gene expression	8
2.2.2	Transcription factors and histone modifications	9
2.2.3	Chromatin Accessibility	10
2.2.4	DNA methylation	12
2.2.5	Chromatin States Segmentation	13
2.2.6	Data Processing	13
2.3	Epigenomics mapping consortia	14
2.4	Other initiatives	16
3	Accessing and analyzing epigenomic data	19
3.1	Accessing epigenomic data	19
3.2	Data analysis tools	20
3.3	Data exploration tools	21
3.3.1	Genome Browsers	21
3.3.2	Local analysis tools developed in the framework of DEEP	22
3.3.3	Web analysis tools in the context of DEEP and BLUEPRINT	23
3.4	Predictive analysis tools	25
3.5	Requirements for large-scale epigenomic data analysis tools	27
4	DeepBlue Epigenomic Data Server	29
4.1	Overview	30
4.1.1	DeepBlue Epigenomic Data Server ecosystem	31
4.2	DeepBlue Server	31
4.2.1	DeepBlue Server Components	32
4.2.2	Scalability	34
4.2.3	Access control	36
4.2.4	Entities	37
4.2.5	Metadata Entities	37
4.2.6	Data Entities	40
4.3	Storing and Retrieving Epigenomic Data and Metadata	42
4.3.1	Storage System	42
4.3.2	Data Model	43
4.3.3	Metadata serialization	43
4.3.4	Storing epigenomic data	45
4.3.5	Genomic data storage	49
4.3.6	Epigenomic data insertion	49
4.3.7	Epigenomic data and metadata retrieval	50

4.4	DeepBlue Populator	51
4.5	Application Programming Interface (API)	54
4.5.1	Entities information operation	56
4.5.2	Listing and searching operations	56
4.5.3	Server-side data processing workflow	58
4.5.4	Data selection	59
4.5.5	Data manipulation	61
4.5.6	Data retrieval	68
4.5.7	Data enrichment	72
4.5.8	Data and metadata insertion and maintenance	75
4.6	Usage examples	76
4.6.1	Identification of <i>TFBSs</i> that overlap <i>H3K4me3</i> peaks and promoter regions	76
4.6.2	Calculating DNA methylation levels across <i>H3K4me3</i> peak regions	78
4.6.3	Enrichment Analysis of DMRs regarding Chromatin States	78
4.6.4	Obtaining gene-specific (epi)genomic information from different tissues	80
4.6.5	Summarize gene expression from hepatocyte experiments	81
4.7	Discussion	82
5	DeepBlue in R/Bioconductor	87
5.1	Overview	87
5.2	Usage examples	88
5.2.1	Data Export	94
5.2.2	Data Caching	95
5.3	Use cases	95
5.3.1	Clustering blood samples by their DNA methylation data	95
5.3.2	Predicting gene expression based on histone marks	97
5.3.3	Constructing cell signatures	98
5.3.4	Metadata and batch effect analysis	102
5.4	Conclusion	112
6	Accessing DeepBlue data from the web-browser	113
6.1	Accessing public epigenomic data	113
6.2	DeepBlue Web Portal	114
6.3	Usage examples	120
6.3.1	Visually selecting and exporting data to a Python script	120
6.3.2	Selecting and downloading specific regions from multiple experiments	122
6.4	Conclusion	125
7	Large scale interactive analysis of epigenomes	127
7.1	Web tools for analyzing epigenomic data	128
7.2	DIVE - Diving in the epigenomic data	129
7.3	Main features	130
7.4	Major features	135
7.4.1	Overlapping regions	135

7.4.2	Sample aggregation	136
7.4.3	Finding similar experiments	137
7.5	Use cases	138
7.5.1	Analyzing the enrichment of chromatin states for finding important transcription factors in cell pluripotency	138
7.5.2	Analyzing transcription factors by overlapping with chromatin states	140
7.5.3	Comparing data from different consortia	142
7.6	Conclusion	145
8	Summary and outlook	147
8.1	Outlook	149
9	Glossaries	153
	List of Abbreviations	153
	Glossary	156
A	Appendix	161
A.1	Data imported by the DeepBlue Server	161
A.2	Implementation details	163
A.2.1	DeepBlue Server	164
A.2.2	DeepBlue Populator	170
A.2.3	DeepBlueR	171
A.2.4	DeepBlue Web Portal	172
A.2.5	DeepBlue Middleware	173
A.2.6	DIVE	173
A.3	Usage example and use cases source code	175
A.3.1	DeepBlue Server	175
A.3.2	DeepBlueR	185
A.3.3	DIVE	201
A.4	DeepBlue Server API	203
A.5	List of publications	212
	References	213

List of Figures

2.1	Levels of chromatin organization.	5
2.2	Histone modifications	6
2.3	Histone modifications	7
2.4	Epigenetic regulation of active gene expression	8
2.5	Sequencing techniques for mapping epigenetic marks	9
2.6	Epigenomic data deluge	10
2.7	Chromatin immunoprecipitation methods	11
3.1	UCSC Genome Browser	22
3.2	deepTools	23
3.3	RnBeads	23
3.4	BLUEPRINT Data Analysis Portal	24
3.5	EpiExplorer	25
4.1	DeepBlue Epigenomic Data Server ecosystem	32
4.2	DeepBlue Server Architecture	33
4.3	DeepBlue Processing Node	35
4.4	DeepBlue Server Cluster Example	36
4.5	Example of the hierarchy of an <i>Uber Anatomy Ontology (UBERON)</i> term .	38
4.6	Example of a DeepBlue Sample	39
4.7	DeepBlue column data types	40
4.8	MongoDB collections with documents	42
4.9	DeepBlue Server Data Model	44
4.10	Example of BioSource document	45
4.11	Example of BED file	46
4.12	Example of BSON Region	46
4.13	Example of BSON region with compressed key	47
4.14	Example of BSON Region	47
4.15	Regions Data Block	48
4.16	DeepBlue processing workflow	59
4.17	Identification of <i>H3K4me3</i> peaks that overlap with promoters and <i>TFBS</i> peaks	77
4.18	Summarizing DNA methylation levels in liver tissue across <i>H3K4me3</i> peaks regions derived from human embryonic stem cells	79
4.19	Summary of gene expression from hepatocyte experiments	83
5.1	Constructed plot containing the retrieved genomic regions divided in two groups by their BioSource	91
5.2	Constructed plot containing the resulting regions	93
5.3	Plotting <i>deepblue_score_matrix</i> command result	94
5.4	Clustering blood samples by their DNA methylation data	96
5.5	Histone marks <i>Area under Curve (AUC)</i> for gene expression prediction . .	98
5.6	Most informative genome location bins for predicting gene expression .	99
5.7	Cell types used for generating cell signatures	100
5.8	Cell types signatures: DNA hypomethylated regions	101

5.9	Cell types signatures: DNA hypermethylated	102
5.10	Fixing metadata encoding and formatting issues	105
5.11	Fixing metadata content with <i>OpenRefine</i>	106
5.12	Ensuring the correctness of the metadata files	107
5.13	Batch analysis tool: selecting experiments	108
5.14	Batch analysis tool: Obtaining detailed metadata content	109
5.15	Batch analysis tool: defining score matrix parameters	110
5.16	Batch analysis tool: interactive <i>Principal Component Analysis (PCA)</i> chart	110
5.17	Batch analysis tool: configuring and executing the batch effect correction	111
5.18	Batch analysis tool: comparison the result	111
6.1	DeepBlue Web Portal: Dashboard	115
6.2	DeepBlue Web Portal: Grid	116
6.3	DeepBlue Web Portal: Full-text search	117
6.4	DeepBlue Web Portal: Data table	117
6.5	DeepBlue Web Portal: Data preview	118
6.6	DeepBlue Web Portal: BioSources hierarchy	119
6.7	DeepBlue Web Portal Grid: selecting analysis data	121
6.8	DeepBlue Web Portal Grid: filtered data	121
6.9	DeepBlue Web Portal Grid: selected data	122
6.10	DeepBlue Web Portal Grid: export selected data	122
6.11	DeepBlue Web Portal: selecting data with the experiments data table	123
6.12	DeepBlue Web Portal Grid: selecting data columns and regions	124
6.13	DeepBlue Web Portal request info interface	124
7.1	DIVE: interface main elements	130
7.2	DIVE suggesting sub-terms when selecting a BioSource	131
7.3	DIVE: Gene Ontology Enrichment	133
7.4	DIVE: LOLA Enrichment Analysis	134
7.5	DIVE: Most dissimilar BioSources	135
7.6	DIVE: Most similar Epigenetic Marks	135
7.7	DIVE: Multiple samples per <i>BioSource</i>	137
7.8	Overview on the Strong Enhancer regions of the <i>h1-hSC</i>	139
7.9	Overlapping chromatin states to POU5F1	140
7.10	Overlapping transcription factors with the chromatin state Strong Enhancer	141
7.11	Overlapping transcription factors with the chromatin state Enh (Enhancer from ENCODE)	142
7.12	Loading a <i>DeepBlue query ID</i> into <i>DIVE</i>	144
7.13	Comparing CREST and DEEP data: overlapping to TSS bivalent regions	145
7.14	Comparing CREST and <i>Deutsches Epigenom Programm (DEEP)</i> data: overlapping to enhancer regions	145
A.1	DeepBlue Server Data Layer	165
A.2	DeepBlue Server integrates different data types	166

List of Tables

2.1	Main histone marks and their effects	7
4.1	Example of DeepBlue Server entities IDs prefixes	41
4.2	The <i>DeepBlue Server</i> API categories and main operations	55
4.3	Description of the <i>DeepBlue Server</i> meta-fields	62
4.4	DeepBlue Server operations to include and maintain metadata and data.	76
4.5	(Epi)genomic data obtained about the FAR1 gene	82
4.6	ROADMAP IDs used for obtaining data on FAR1 gene	83
7.1	Transcription factors selected for the overlap analysis using chromatin states	141
A.1	Genome assemblies available in the DeepBlue Server	161
A.2	Gene models available in the DeepBlue Server	161
A.3	Experiment files available in the DeepBlue Server	161
A.4	Annotation files available in the DeepBlue Server	162
A.5	Annotation files available in the DeepBlue Server	162
A.6	Chromatin States used in the ROADMAP project	163
A.7	Chromatin States used in the ROADMAP project	163
A.8	Chromatin States used in the ENCODE project	164
A.9	Index overhead in different region blocks size	169
A.10	Lua commands available in the <i>DeepBlue Server</i>	171

4.1	Example of the <i>faceting_experiments</i> operation	57
4.2	Example of the <i>search</i> operation	58
4.3	Example of the <i>filter_regions</i> operation	63
4.4	Example of the <i>filter_by_motif</i> operation	64
4.5	Example of <i>find_motif</i> and <i>intersection</i> operations	64
4.6	Example of <i>find_motif</i> and <i>overlap</i> operations	65
4.7	Example of <i>select_genes</i> , <i>flank</i> , and <i>merge</i> operations	66
4.8	Example of <i>aggregate</i> and <i>filter</i> operations	67
4.9	Example of <i>get_request_data</i> operation	68
4.10	Example of <i>select_experiments</i> and <i>count_regions</i> operations	68
4.11	Example of the <i>binning</i> operation	69
4.12	Example of the <i>coverage</i> operation	69
4.13	Example of the <i>distinct_column_values</i> operation	70
4.14	Example of a workflow obtaining ChIP-seq data in a tabular format using the <i>select_regions</i> operation	71
4.15	Example of the <i>score_matrix</i> operation	72
5.1	Command <i>search</i> example.	89
5.2	Commands <i>select_experiments</i> and <i>count_regions</i> example.	89
5.3	Example of selecting, filtering, and downloading epigenomic data	90
5.4	Connecting downloaded regions to <i>GViz</i> library.	91
5.5	Example of summarizing regions and displaying the result.	92
5.6	Example of building a score matrix.	93
5.7	Example of exporting (epi)genomic data to the local hard disk	94
5.8	Example of exporting metadata.	94
5.9	Example of storing multiple requests data.	95
A.1	Auxiliar function for downloading workflow data.	175
A.2	Usage Example 1: Identification of <i>H3K4me3</i> peaks that overlap with promoters in all BLUEPRINT datasets and subsequent identification of transcription factor peaks that overlap with these promoters in all ENCODE datasets.	176
A.3	Usage Example 2: Summarizing DNA methylation levels in liver tissue across <i>H3K4me3</i> peaks regions derived from human embryonic stem cells.	177
A.4	Usage Example 3: Enriching hypomethylated regions via Chromatin States information.	179
A.5	Usage Example: Comparing the <i>FAR1</i> gene (epi)genomic information in liver and brain tissues.	182
A.6	Usage Example: Summarize genes expression from Hepatocytes experiments.	184
A.7	Usage example: Study DNA methylation level across 206 BLUEPRINT blood samples. Data acquisition.	185
A.8	DeepBlueR use case: predicting gene expression from histone data (loading the data)	188

A.9 DeepBlueR use case: predicting gene expression from histone data (performing predictions)	192
A.10 Generating biosource signatures for biomarkers identification: main file	193
A.11 Generating biosource signatures for biomarkers identification: retrieving the data	196
A.12 Generating biosource signatures for biomarkers identification: generating signatures	198
A.13 Generating biosource signatures for biomarkers identification: displaying results	199
A.14 Generating biosource signatures for biomarkers identification: testing calculation	200
A.15 Pre-processing data for comparing data from <i>DEEP</i> and CREST in <i>DIVE</i>	201

Introduction

In 2001, two drafts of the human genome sequence first became available (Lander *et al.* 2001; Venter *et al.* 2001), holding the promise of quickly advancing our understanding of gene regulation and disease mechanisms. However, researchers have soon realized that they have to look beyond the genome to understand the complex behavior observed between different cell types that all share the same genetic information. Epigenetic research focuses on understanding factors influencing gene regulation that are not coded in DNA and yet some of which are heritable. The field covers various cellular mechanisms such as DNA methylation, histone modification, RNA function, and transcription factor binding sites. Recent advances in high-throughput profiling technologies allow for systematically collecting data on each of these mechanisms in large-scale experiments (P. J. Park 2009; Tsompana and Buck 2014; Y. Li and Tollefsbol 2011).

These efforts are fostered and concerted by international collaborations, such as the *International Human Epigenome Consortium* (IHEC) (Stunnenberg *et al.* 2016) and its member projects', such as *BLUEPRINT Epigenome Project* (Adams *et al.* 2012; Martens and Stunnenberg 2013), *DEEP* (<http://www.deutsches-epigenom-programm.de>), *NIH Roadmap Epigenomics Mapping Consortium* (REMC) (Roadmap Epigenomics Consortium *et al.* 2015), and *The Encyclopedia of DNA Elements* (ENCODE) (ENCODE Project Consortium 2012). As a result of these collaborations, researchers can exploit massive amounts of publicly available epigenomic¹ data on dozens of cell types, cell lines and tissues. Access to these data is streamlined by existing data portals (Bujold *et al.* 2016; J. M. Fernández *et al.* 2016; Sloan *et al.* 2015) and, in principle, allows for answering critical biomedical questions. However, few researchers possess the necessary technical skills and time to operate on such large datasets.

Moreover, working with such data requires a suitable computational infrastructure not accessible ubiquitously. The difficulties for accessing the publicly available epigenomic data creates a severe bottleneck in research and, as a result, data from these costly experiments are currently underused. To address the current epigenomic data deluge, the *DeepBlue Epigenomic Data Server* and its companion libraries and analysis software, such as *DIVE* were developed.

The *DeepBlue Epigenomic Data Server* and its ecosystem provide straightforward access to the public epigenomic data produced by the *IHEC* member projects' and reprocessed by the *ChIP-Atlas* (Oki *et al.* 2018a). Access to *DeepBlue* is facilitated through a powerful API, a data portal, and a data analysis software. The API does not only allow for data access but also allows performing complex operations on epigenomic data directly on

¹ In this thesis, the term “epigenetic” describes the changes, characteristics, and mechanisms, while “epigenomic” refers to the collective of epigenetic events in a specific entity, e.g. a single cell, a cell line, a cell type, tissue, organ, or organism). For this reason, the term “epigenomic data” refers to the epigenomic information to a specific epigenetic entity.

the server. It empowers researchers to perform large-scale data analysis without the need for sophisticated hardware and software.

1.1 Thesis goals and outline

This thesis seeks to achieve three goals:

- Organize the public epigenomic data such that researchers can answer their questions more easily
- Provide means for finding, operating, and analyzing the existing and currently generated epigenomic data in a way that matches the current pace of epigenomic data generation
- Empower researchers with methods and techniques for performing large-scale epigenomic data analysis

The *DeepBlue Epigenomic Data Server* and its ecosystem were developed with the intent to answer the previous goals. The result was an ecosystem of tools and libraries empowering users to finding, retrieve, manipulating, and analyzing public epigenomic data.

This thesis is divided into eight chapters plus the appendices. It is structured as follows:

- Chapter 1 introduces into the topic.
- Chapter 2 presents the necessary background for understanding current epigenomic data analysis questions. It presents the main data producers, commonly used data formats, and major access and analysis issues.
- Chapter 3 introduces the reader to methods and software for storing, analyzing, and exploring epigenomic data. This chapter also presents the requirements for large-scale epigenomic data analysis tools.
- Chapter 4 presents the *DeepBlue Epigenomic Data Server*. It presents its main concepts, and how it is structured, how the data is stored, its API, and its main methods.
- Chapter 5 presents the *R/Bioconductor* package *DeepBlueR*, a package that allows users to perform data analysis in the *R/Bioconductor* environment.
- Chapter 6 presents the *DeepBlue Web Portal*, a web data portal where users can easily find and download the needed epigenomic data.
- Chapter 7 presents *DIVE*, a powerful web application that allows users to perform large-scale epigenomic data analysis directly in their web browser.
- Chapter 8 gives perspectives and outlook on the exciting field of computational epigenetics and offer insights into how the methods and tools developed in this thesis may help to elucidate the pressing epigenomic questions.
- In the appendix, Chapter A provides lists the data imported into the *DeepBlue Server*, implementation details of the *DeepBlue Epigenomic Data Server* components, the usage examples and use cases source code, and the *DeepBlue Server* API.

2

Computational Epigenetics

This chapter introduces the reader to the main aspects of epigenetics and epigenomics. Section 2.1 provides a general introduction to epigenetics, its regulatory mechanisms, and elements. Section 2.2 describes epigenomic data, how they are obtained, and the data volume growth. Section 2.3 presents the epigenome mapping consortia and projects for reprocessing the public available data.

2.1 Epigenetics

The sequencing of the human genome which yielded two first drafts of the whole genome, one of them, publicly available in February 2001 came to be a cornerstone in the field of life sciences (Lander *et al.* 2001). Scientists believed to have found the answer to how human cells are programmed. But in the following years, it has become clear that genes are not the only factors to regulate the many different functions of the human body. Virtually all cells in our body contain the same DNA, but how do humans have more than 200 different cell types? How is it possible that each cell type has a different molecular and phenotypic make up and gene expression profile?

Through the study of epigenetics such question can be answered. The term *epigenetics* is composed of the word “genetics” and the Greek prefix “epi” meaning “in addition to”, in other words, an additional layer of information and control to the existing genomic information.

Epigenetics deals with the mechanisms in our cells that regulate the expression of our genes. Virtually¹, each single cell contains the complete ‘genetic recipe’ of an individual human being. Given the high degree of specialization of different cell types, only a part of the genetic information is read and encoded to proteins.

Epigenetics can be explained through an analogy: the DNA is a ‘big recipe book’, containing genes, recipe, of each protein. The recipe book is the same across all cells of an organism, but the way it is read and interpreted depends on each restaurant. Our recipe book contains recipes of different traditional food styles: Indian food, Italian food, German food, and so on. However, some recipes are prepared in only some specific restaurants, the same applies to the genes that their expression depends on their cell type. So, epigenetics controls how and which genes are expressed in each cell of our body.

Goldberg *et al.* 2007 gives a more formal definition of epigenetics as “*the study of any potentially stable and, ideally, heritable change in gene expression or cellular phenotype that occurs without changes in Watson-Crick base-pairing of DNA*”. It means that modifications in the

¹ As an exception, erythrocytes (red blood cells) do not carry genetic material

epigenomic structure are reflected by the gene expression and in the cellular phenotype, but not directly in the DNA sequence.

Epigenetics also plays a fundamental role in cell differentiation. The control of the genes being switched on or off depends directly on the epigenetic mechanisms of the cell. These mechanisms depend on the cell memory that is “stored” using epigenetic marks, such as DNA methylation. The influence or the changes of the epigenetic marks in the cell differentiation are a well studied subject in haematopoiesis (Farlik *et al.* 2016) and embryonic development (Sarmiento *et al.* 2004).

In the recent past, it has become evident that epigenetic programming is closely linked to human health. Epigenetic aberrations have significant implications for widespread diseases like diabetes, mental disorders, and cancer. When it comes to understand how and why changes in gene expression occur, scientists and physicians are one step closer to provide better treatments and prevent those diseases. Thus, customized treatments can be offered according to the epigenetic profile of the individual patient. This would be a breakthrough in personalized medicine, a clinical approach which has proven very promising in recent years.

2.1.1 Epigenetic Elements

The DNA in the cells is tightly packed. If unfolded and stretched, the total length of DNA of each human cell, with all 6.4 billion basepairs (bps), would be roughly two meters. Due to its many levels of packing, the DNA fits in the cell nucleus. Figure 2.1 shows the different levels of chromatin organization, starting from the beads-on-a-string in the nucleosomes until the chromosomic conformation during the cell division.

The regulation of gene expression is closely connected to how the DNA is packaged inside the cell nucleus. Martens and Stunnenberg 2013 summarize: “this packaging is orchestrated via chromatin, the complex of DNA, RNA and proteins that provides functionality to the genome”. In contrast to the DNA, the chromatin is highly dynamic, with multiple changes happening simultaneously in different places and levels during the cell life cycle. The chromatin plasticity is achieved by the methylation of cytosines in the DNA, process named as DNA methylation, and by modification in the chemical structure of the histones that compose part of the nucleosomes, the histone marks (or modifications)².

Histone Marks

Epigenetic marks are modifications in the chromatin that control the gene expression of the cell and consequently, its phenotype. Figure 2.2 illustrates these epigenetic elements and how they control the chromatin structure and gene accessibility.

The nucleosome core particle consists of approximately 146 bp of DNA wrapped around a histone octamer, consisting of eight histones, two copies of each *H2A*, *H2B*, *H3*, and *H4*. The histone *H1* and its variants influence the chromatin compaction. These histones are subject to chemical modifications (Figure 2.3) which, as previously mentioned, directly influence the chromatin conformation, and thus the gene expression.

² In this thesis, the terms *histone marks* and *histone modification* are used interchangeably.

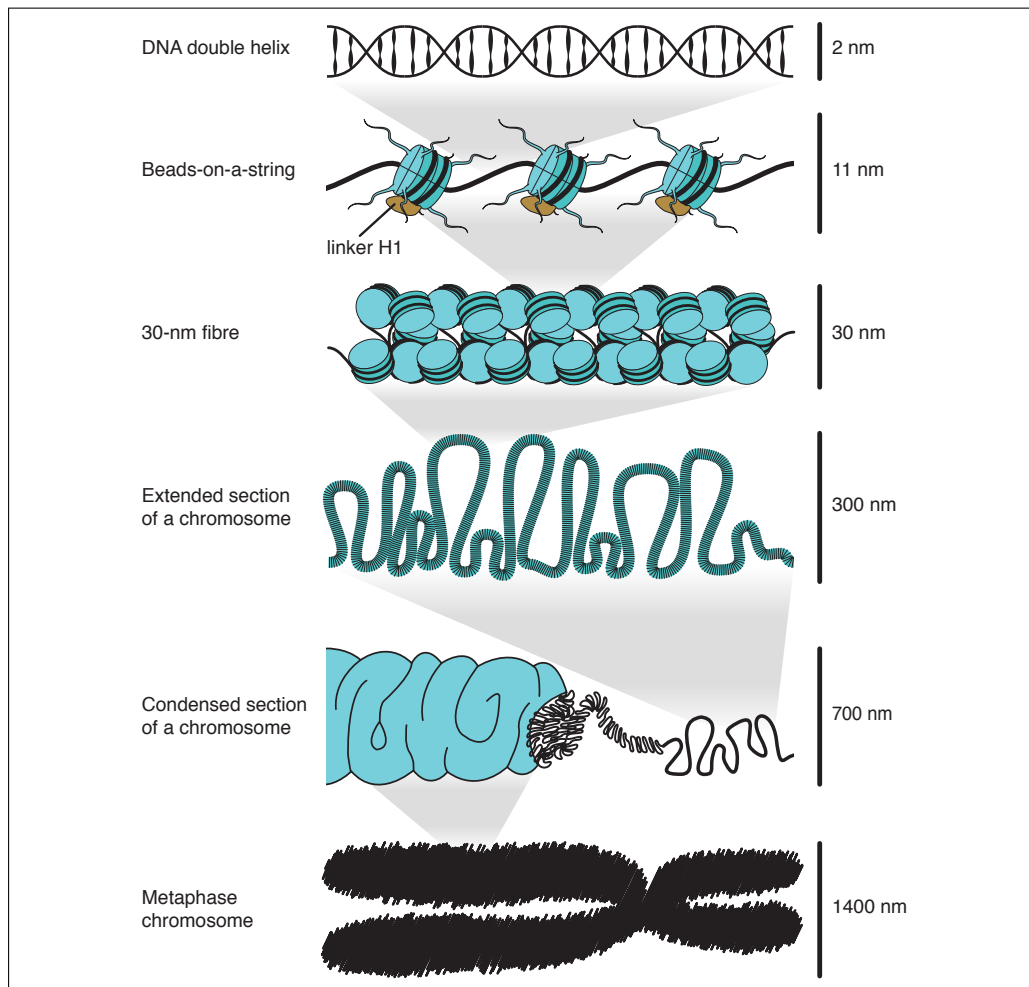


Figure 2.1: Levels of chromatin organization. Reprinted from Müller 2017.

The chromatin conformation influences directly the gene expression level: when the chromatin is *open* the gene can be expressed. Hence, by modifying the chromatin conformation, histone modifications can generate synergistic or antagonistic interactions on gene expressions, strongly affecting organism traits.

Histone modifications control the chromatin state, turning genomic regions transiently active or silent. The pattern in which the histone modifications influence gene expression, is named “histone code”. It is proposed that this epigenetic marking system represents a fundamental regulatory mechanism that has an impact on most, if not all, chromatin-template processes, with far-reaching consequences for cell fate decisions in both normal and pathological development (Jenuwein and Allis 2001). It is still in discussion if the histone modifications are causative or just correlations (Calo and Wysocka 2013). Table 2.1 summarizes the general effects of the most well-known histone marks.

DNA methylation

DNA methylation is an epigenetic modification by which a methyl group is added to cytosine or adenine in the DNA molecule. DNA methylation plays a fundamental role in regulating gene expression and therefore a broad range of biological processes and

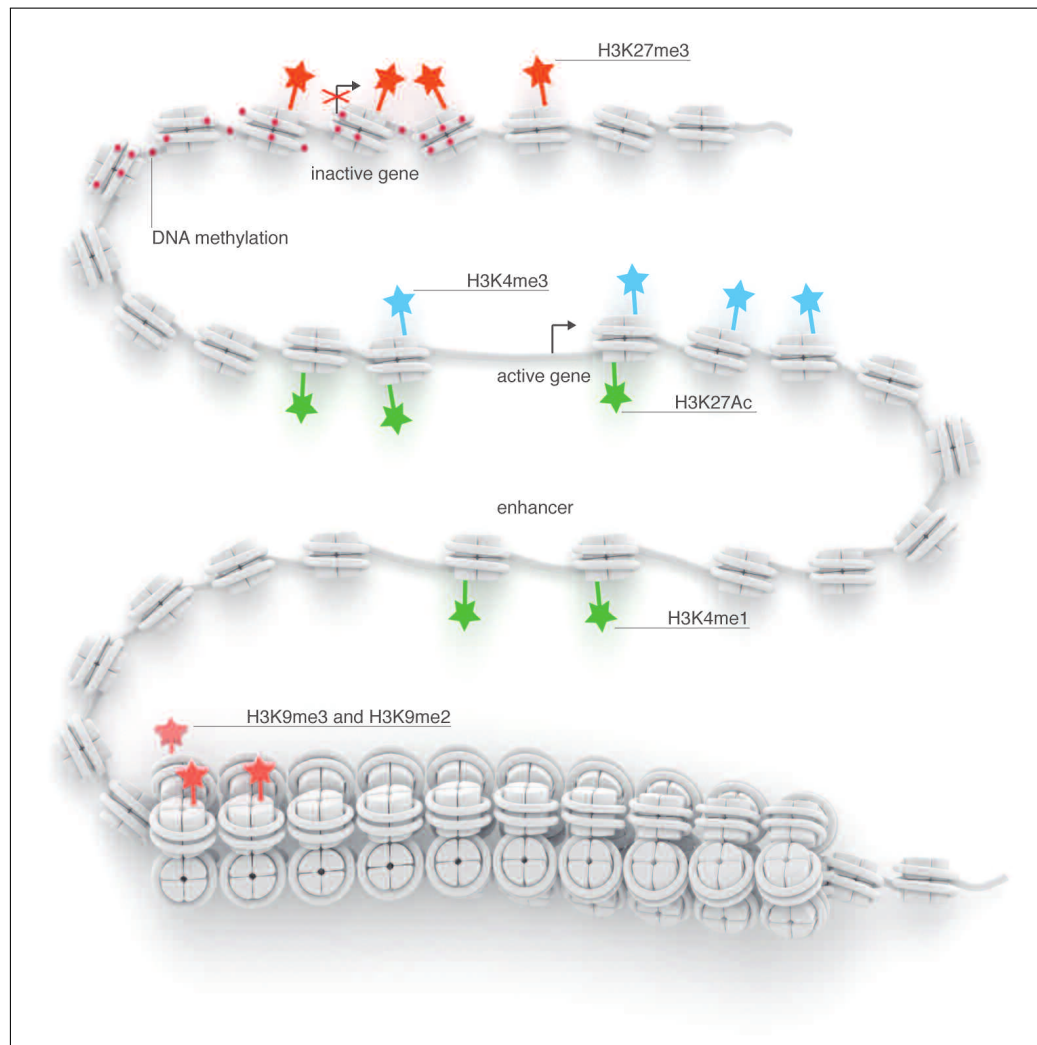


Figure 2.2: Schematic overview of epigenetic modifications and effects on chromatin structure and accessibility of genes. Reprinted from Martens and Stunnenberg 2013. Obtained from the Haematologica Journal website <http://www.haematologica.org>.

diseases (Yong *et al.* 2016), like genomic imprinting (E. Li *et al.* 1993), x-chromosomal inactivation (A. J. Sharp *et al.* 2011), repression of transposable elements (Ikeda and Nishimura 2015), cell differentiation (Khavari *et al.* 2010), aging (Ciccarone *et al.* 2018), and carcinogenesis (Nakajima *et al.* 2008; Akhavan-Niaki and Samadani 2013).

This epigenetic modification usually occurs in regions with a high abundance of CpG dinucleotides, named CpG islands. When CpG islands are located in a gene promoter, DNA methylation typically acts to repress gene transcription. In mouse and human, around 60–70% of genes have a CpG island in their promoter region and most of these CpG islands remain unmethylated independently of the transcriptional activity of the gene, in both differentiated and undifferentiated cell types (Weber *et al.* 2007)

Figure 2.4 provides an overview of the epigenetic regulation of active gene expression: the *RNA polymerase II* (RNAPII) complex binds to the promoter region of a gene. Active promoters frequently carry the $H3K4me3$ histone modification as well as low levels

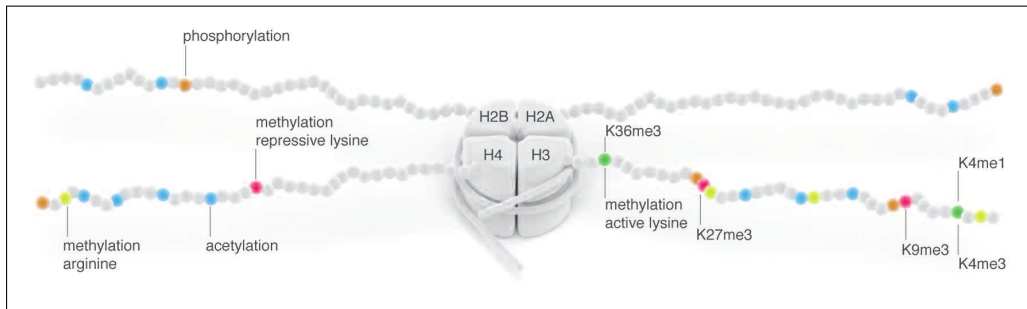


Figure 2.3: Schematic overview of histone modifications. Reprinted from Martens and Stunnenberg 2013. Obtained from the Haematologica Journal website <http://www.haematologica.org>.

Histone Mark	Effect	Reference
H3K27me3	facultatively repressed gene	Barski <i>et al.</i> 2007
H3K27ac	distinguishes active from poised enhancers	Creyghton <i>et al.</i> 2010
H3K4me1	actively transcribed promoters	Heintzman <i>et al.</i> 2007
H3K4me2	actively transcribed promoters	Pekowska <i>et al.</i> 2010
H3K4me3	actively transcribed promoters	Koch <i>et al.</i> 2007
H3K9me3	constitutively repressed genes	Barski <i>et al.</i> 2007
H3K36me3	actively transcribed gene bodies	Pu <i>et al.</i> 2015
H3K9ac	actively transcribed promoters	Gates <i>et al.</i> 2017
H3K14ac	actively transcribed promoters	Karmodiya <i>et al.</i> 2012

Table 2.1: Main histone marks and their effects.

of DNA methylation and are associated with nucleosome-depleted regions. Transcriptional elongation is associated with *H3K36me3* and DNA methylation in gene bodies. Active enhancers (usually marked by histone acetylation and *H3K4me1*) can be located dozens of kilobases upstream or downstream of a gene and come in close proximity to promoter regions via loop structures. Transcription factors (TFs) can localize in promoter regions or distal elements in order to regulate the assembly of the transcriptional machinery (Müller 2017).

2.2 Epigenomic Data

Next Generation Sequencing (NGS) are technologies capable of high-throughput sequencing millions of short DNA reads and revolutionized the methods of producing epigenomic data. Sequencing costs have decreased significantly, from billions to a few thousand dollars (National Human Genome Research Institute 2016), and sequencing the entire genomes, from bacteria to mammals, became a common task in research (Metzker 2010; Goodwin *et al.* 2016).

Development and improvements in the sequencing protocols extended their use beyond the scope of genomes, leading to the improvement of epigenome mapping techniques. For example, in the last decade, technologies for mapping different epigenetic marks were developed (Figure 2.5).

Due to these new technologies, thousands of epigenomic data files are regularly being generated and available for researchers for studying epigenomic regulation processes.

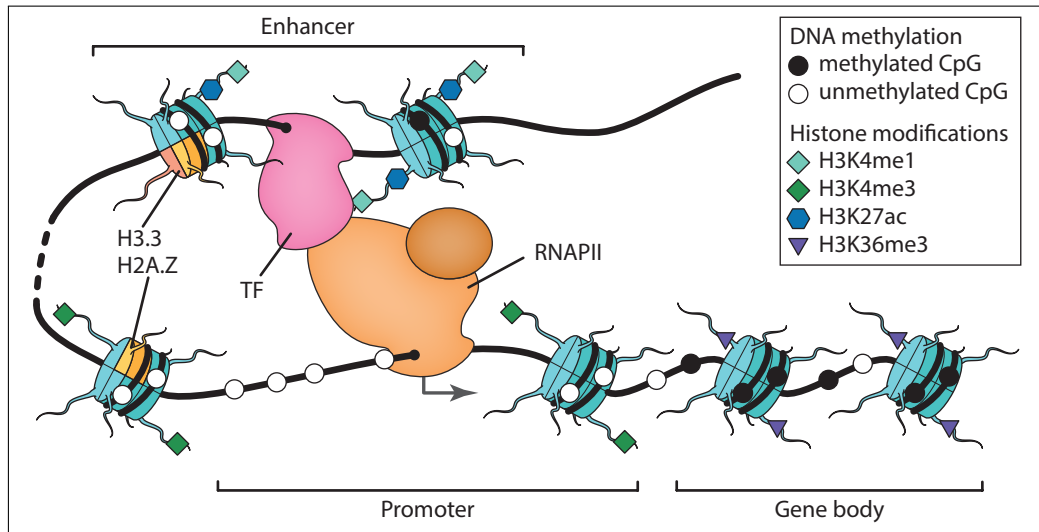


Figure 2.4: Epigenetic regulation of active gene expression: *RNAPII* complex binds the promoter region of a gene. Active promoters frequently carry the *H3K4me3* histone modification as well as low levels of DNA methylation and are associated with nucleosome depleted regions. Transcriptional elongation is associated with *H3K36me3* and DNA methylation in gene bodies. Active enhancers (marked by histone acetylation and *H3K4me1*) can be located dozens of kilobases upstream or downstream of a gene and contact promoter regions via loop structures. TFs can localize in promoter regions or distal elements in order to regulate the assembly of the transcriptional machinery. (Reprinted from Müller 2017).

In 2013, roughly 2,000 sequencing instruments in laboratories and hospitals around the world collectively sequenced 15 quadrillion nucleotides per year, which equals about 15 petabytes of compressed genetic data (Schatz and Langmead 2013). The amount of produced data increased with the emergence of microarray and NGS technologies. These DNA sequencing techniques are used in the epigenomic mapping assays for generating the epigenomic data.

As shown in Figure 2.6, during the year of 2003, less than 1 gigabyte of processed epigenomic data were openly available. In 2009, this number exceeded 1 terabyte. By 2015, the amount of this type of data grew by a factor 20. Nowadays, the amount of data is almost doubling each year, reaching almost 50 terabytes in 2017, and it is expected to reach 80 terabytes in 2019.

The following section presents the main technologies for mapping epigenetic marks.

2.2.1 Gene expression

Different *RNA-sequencing* (*RNA-seq*) techniques are employed for quantifying gene expression. These techniques use the reverse transcriptase enzyme for converting the RNA to *complementary DNA* (*cDNA*) (Telesnitsky and Goff 1997) from which a sequence library is built. Different protocols are necessary for obtaining the RNA from different regions of the cells: nucleus, cytosol, or the entire cell.

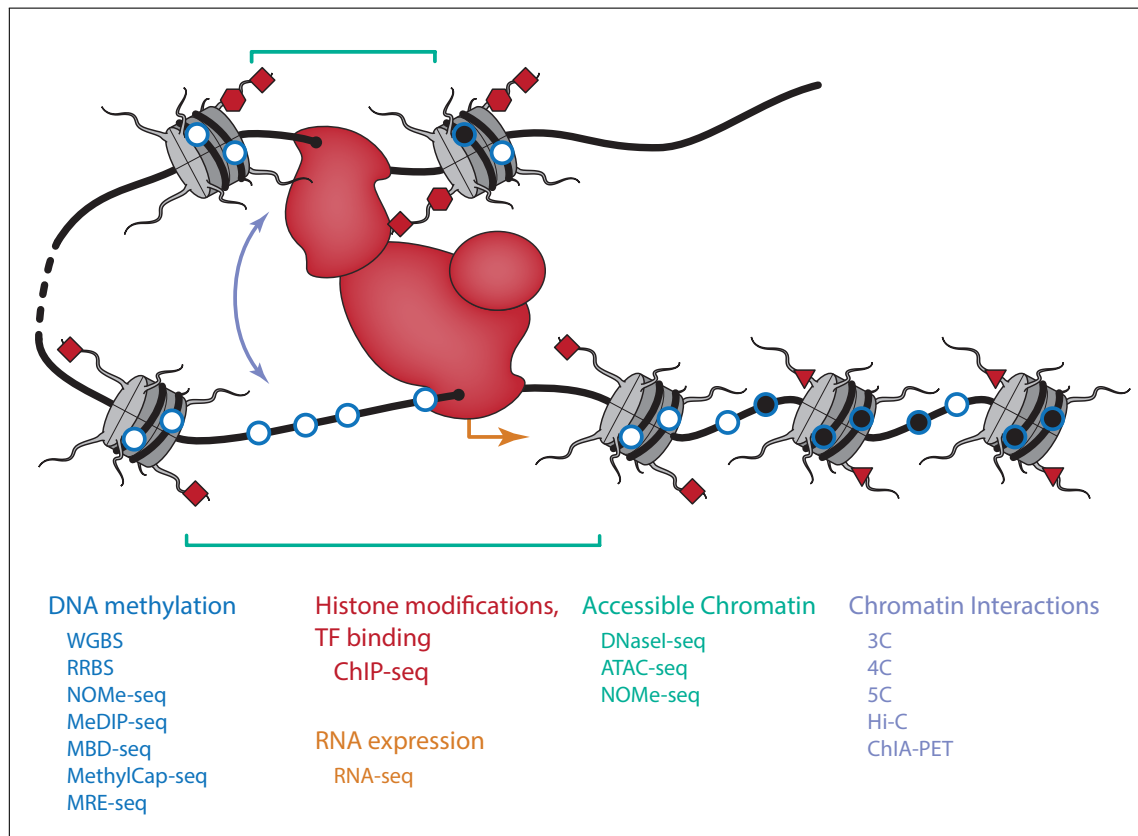


Figure 2.5: Sequencing techniques for mapping epigenetic marks. Colors indicate different epigenetic marks and the corresponding techniques (Reprinted from Müller 2017).

The RNA reads are then mapped to a reference genome, followed by identifying the genes and isoforms, and quantifying their expression level using the number of reads mapped to each gene and its isoform. Garber *et al.* 2011 present a selection of tools and pipelines for gene expression quantification.

2.2.2 Transcription factors and histone modifications

Chromatin Immunoprecipitation (ChIP) followed by sequencing is a technique for genome-wide profiling of DNA-binding proteins, histone modifications or nucleosomes (P. J. Park 2009). A typical *Chromatin Immunoprecipitation-Sequencing (ChIP-seq)* experiment results in a map of the histone modification or transcription factors across the entire genome. *ChIP-seq* involves crosslinking DNA-binding proteins to DNA by treating cells with formaldehyde and fragmenting chromatin by sonication or enzymatic digestion. Immunoprecipitation of the crosslinked chromatin is performed using an antibody that recognizes a specific transcription factor or histone modification, which results in the identification of binding sites in the genome for the factor or histone modification of interest. After purification of the precipitated fragments, the sample can be analyzed by *Polymerase Chain Reaction (PCR)* to study particular genes. *ChIP-seq* can be performed genome-wide (Farnham 2009). Figure 2.7 presents an overview of *ChIP-seq* methods.

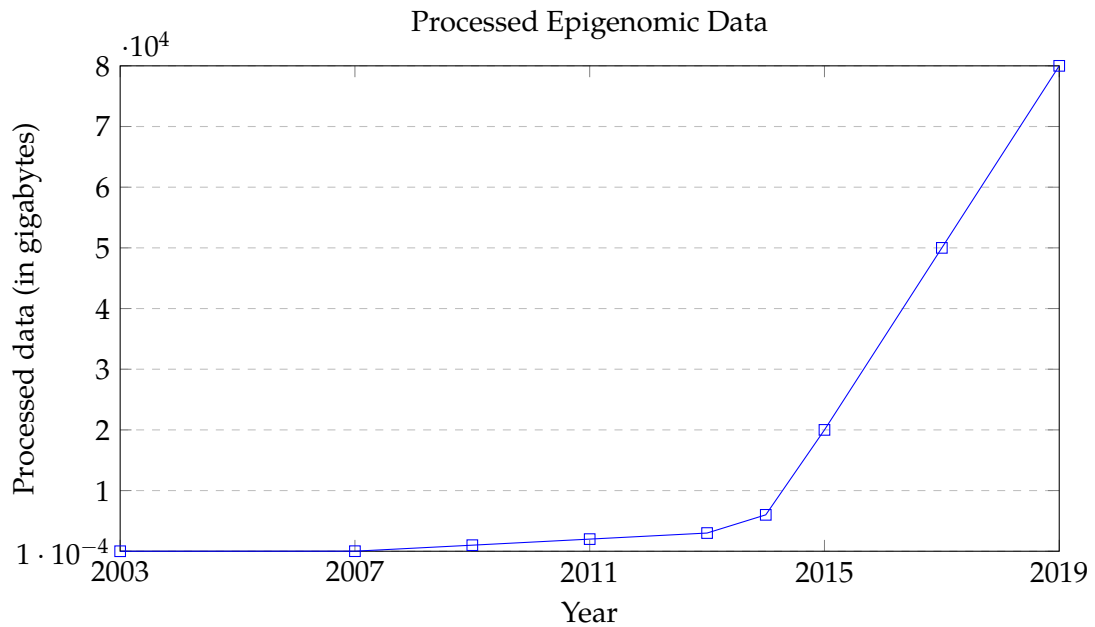


Figure 2.6: Epigenomic data deluge: less than 1 gigabyte of processed epigenomic data were openly available in 2003. Only in 2009, this number exceeded 1 terabyte. It was close to 50 terabytes in 2017, and it is expected to reach 80 terabytes in 2019.

After the amplified sequence fragments are sequenced, the result is stored in a *FASTQ*³ file, and the sequence fragments are mapped to a reference genome using a genome alignment tool (Escalona *et al.* 2016) resulting in a *Binary Alignment/Map* (BAM) file⁴. With the BAM file, the signal strength is calculated using the count of fragments in a given genomic location. This signal can be normalized using the input signal, which is obtained using an analogous sequencing experiment, using the same biological material, but not using the immunoprecipitation step. Then, the signal data is stored using the *Wiggle Track Format* (WIG)⁵ file format.

Finally, peak calling software (Y. Zhang *et al.* 2008; Heinig *et al.* 2015) identifies genome regions where the signal has statistical significance. These software methods vary in their distributional assumptions and employed statistical models (Koohy *et al.* 2014; R. Thomas *et al.* 2016). The peak regions can be stored using the *Browser Extensible Data* (BED)⁶ format or some other custom column-based file format.

2.2.3 Chromatin Accessibility

Identifying DNA regions accessible to the transcription machinery is crucial for the characterization of the gene expression regulatory process. Chromatin accessibility approaches measure the effect of chromatin structure modifications on gene transcription, in contrast to using *ChIP-seq* for measuring histone occupancy (Tsompana and Buck

³ <http://maq.sourceforge.net/fastq.shtml>

⁴ <https://www.ncbi.nlm.nih.gov/sra/docs/submitformats/#bam-files>

⁵ <https://genome.ucsc.edu/goldenpath/help/wiggle.html>

⁶ <https://genome.ucsc.edu/FAQ/FAQformat.html#format1>

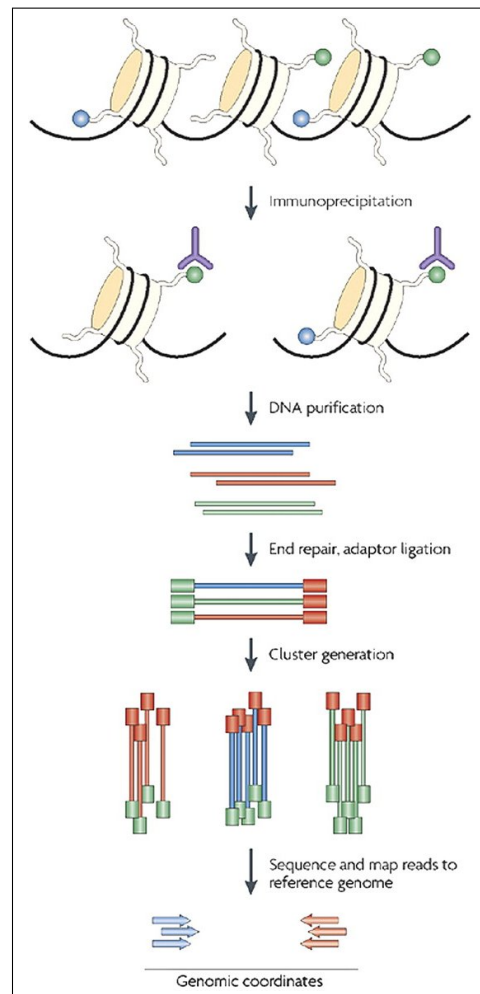


Figure 2.7: Genome-wide mapping of histone modifications and other DNA–protein interactions has relied on next generation sequencing (NGS) technologies (ie, ChIP-Seq) which have provided more precise and comprehensive landscapes of histone modifications in the entire genome. (Reprinted and modified from Ku *et al.* 2011).

2014). Currently, the four most used techniques for quantifying chromatin accessibility are:

DNase-Seq (*DNase-seq*) uses the deoxyribonuclease I (DNase I) (Weintraub and Groudine 1976; Song and Crawford 2010; John *et al.* 2013) for cutting the DNA in its open regions. It is a well established but laborious technique and requires many cells (1 to 10 million).

Formaldehyde-assisted isolation of regulatory elements (*FAIRE-seq*) is based on the phenol-chloroform separation of nucleosome-bound and free sonicated areas of a genome, in the interphase and aqueous phase, respectively. It is a simpler technique, already applicable using fewer cells (100,000 to 10 million) but the low-signal-to-noise ratio can make the data interpretation very difficult (Giresi *et al.* 2007; Giresi and Lieb 2009; J. M. Simon *et al.* 2012; J. M. Simon *et al.* 2013).

Assay for Transposase-Accessible Chromatin using Sequencing (ATAC-seq) is a simpler protocol than *DNaseI-seq*, requiring fewer cells (500 to 500,000) but the cells must be freshly isolated. It has the benefit of mapping open chromatin, transcription factors, and nucleosome occupancy in the same protocol execution (Buenrostro *et al.* 2015; Cusanovich *et al.* 2015).

Nucleosome Occupancy and Methylome-Sequencing (NOMe-seq) is a different technique from the previously presented. The input DNA is treated with a GpC Methyltransferase (M.CviPI) that specifically methylates cytosines in GpC context in regions of accessible chromatin. The *NOMe-seq* technique also allows for quantification of the DNA methylation levels for the GpCs in the same protocol execution (Kelly *et al.* 2012).

Similar to *ChIP-seq* protocols, peak callers are used for identifying open chromatin regions (Koohy *et al.* 2014). These regions are stored in a *BED*-like format, where it is possible to obtain the accessible genomic regions for analyzing open chromatin data.

2.2.4 DNA methylation

Sequencing-based methods for profiling DNA methylation can be broadly categorized into protocols based on selective enrichment, specific cleavage by restriction enzymes or treatment with sodium bisulfite (Müller 2017). Benchmarking studies show that there is generally high agreement between the various protocols (Bock *et al.* 2010; R. A. Harris *et al.* 2010). An extensive list of reviews about DNA methylation quantification are available at Parle-Mcdermott and Harrison 2011; Bock 2012; Dahl and Guldborg 2003; Yong *et al.* 2016; Kurdyukov and Bullock 2016. Teschendorff and Relton 2018 describe techniques for integrating DNA methylation data with other epigenetic marks.

For the scope of this work, most frequently used DNA methylation mapping and profiling techniques⁷ are presented:

Bisulfite sequencing is a technique in which DNA is first treated with bisulfite that converts cytosines to uracil but leaves 5-methylcytosine unaffected, leaving only methylated cytosines. It is used for measuring the DNA methylation level and estimates the methylation ratio rather than enrichment levels. It is the gold-standard technology for detection of DNA methylation because it provides a qualitative, quantitative and efficient approach to identify 5-methylcytosine at single base-pair resolution (Y. Li and Tollefsbol 2011).

Whole Genome Bisulfite Sequencing (WGBS) measures the ratio of methylated and unmethylated molecules across the whole genome. In general, this method yields the best results but requires resequencing the entire genome multiple times for every experiment (Stevens *et al.* 2013). It is the most desirable technique but requires many resources, being not suitable for projects with many samples.

Reduced Representation Bisulfite Sequencing (RRBS) is similar to WGBS, but it uses restriction enzymes to limit the enrichment and analysis of specific genomic

⁷ Array methods are not presented because they are not supported by the *DeepBlue Epigenomic Data Server*

regions that have a high CpG content, reducing the amount of DNA needed to be sequenced, thus, reducing the final cost (Meissner *et al.* 2005).

- Whole genome shotgun bisulfite sequencing (WGSBS)* is a technique similar to WGBS but using bisulfite with shotgun sequencing, where the DNA is broken up into short fragments which are then sequenced in parallel (Anderson 1981).
- NOMe-seq* is used for quantifying the DNA methylation rate and also the chromatin accessibility. Its output contains the chromatin accessibility and DNA methylation values in the fraction of methylated and unmethylated cytosines (Lay *et al.* 2018).
- MeDIP-seq/MRE* is a combination of the *Methylated DNA Immunoprecipitation Sequencing (MeDIP-seq)* (Weber *et al.* 2005) with the *methylation-sensitive restriction enzyme digestion (MRE)* (Maunakea *et al.* 2010) technique. The data from these experiments is processed by specific software, such as the *methylMnM* package (B. Zhang *et al.* 2013) and tool *methylCRF* (Stevens *et al.* 2013) that reports fractional methylation in the range from 0 to 1 (D. Li *et al.* 2015).

The processed DNA methylation data files are divided into two main groups: data files with the DNA methylation level across the genome or the regions of interest, and files containing the hypomethylated and hypermethylated regions. The DNA methylation level data files usually are stored in *WIG* or *BED*-like data files and are accompanied with metadata files reporting the coverage (how many times a genomic region was sequenced). The hypo/hyper-methylated data is derived from the previous files using specific tools (Assenov *et al.* 2014). Such derived data contains disjoint genomic regions with their methylation levels and is stored in *BED*-like files.

2.2.5 Chromatin States Segmentation

As previously explained, the chromatin structure and its accessibility are strongly controlled by histone modifications. These epigenetic marks are co-located in the genome, making it necessary to computationally summarize the pattern of multiple histone marks and their implication on the genomic region, to segment the genome into meaningful biological units, each one being associated with a chromatin state.

Tools like *ChromHMM* (Ernst *et al.* 2011; Ernst and Kellis 2012; Ernst and Kellis 2017) were developed to perform such tasks. They segment the genome into regions of varying chromatin states using a multivariate *Hidden Markov Model (HMM)* that explicitly models the observed combination of histone marks (Ernst *et al.* 2011). The used technique, named *Chromatin State Segmentation (CSS)*, defines chromatin states based on different combinations of histone modifications, and associates each different genomic regions with a specific chromatin state. CSS is broadly used in epigenomic data analysis. For facilitating further data analysis interpretation, Appendix A.1 provides tables with the chromatin states provided by the BLUEPRINT, REMC, and ENCODE datasets.

2.2.6 Data Processing

It is plausible to classify the epigenomic data into *raw* and *processed data*. The *raw* data is the data generated from the laboratory instruments, such as, the DNA sequencing

machines. Due to its complexity, bias, and quality issues, the *raw* data cannot be directly analyzed. Rather, it is necessary to be pre-processed first.

The epigenomic data pre-processing is composed of sequential steps, creating a data flow similar to a pipeline or workflow⁸. It is possible to build such data processing pipelines by developing simple programming scripts which afford the communication between the data processing tools. For facilitating the data processing, tools like Galaxy (Giardine, Riemer, Hardison, Burhans, Elnitski, Shah, Y. Zhang, Blankenberg, Albert, Taylor, *et al.* 2005a; Goecks *et al.* 2010; Blankenberg *et al.* 2010) and Taverna (Wolstencroft *et al.* 2013) provide intuitive interfaces, but they also require a complicated initial set-up.

Ebert *et al.* 2015 propose a simple and straightforward approach, where each pipeline step is described in a *XML* file. The description contains fundamental information like the software name, version, and input (*raw* data files and configurations) and output files (*processed data* and metadata). This approach also performs automatic consistency checks and metadata handling, where the *XML* files containing the processing steps are stored and available together with the metadata and processed files.

The processed epigenomic data is typically stored into *WIG* and *BED* files. *WIG* files contain the signal data, for storing contiguous regions of the genome, across the entire genome. Due to the amount of information, a single *WIG* file of a *WGBS* experiment requires 2 to 3 gigabytes of disk space.

Due the size of the *WIG* files, the *bigWig* is a binary format developed for storing the signal data. The *bigWig*⁹ file is smaller than its counterpart *WIG*, and it has an index, facilitating quick retrieval of the signal from specific genomic locations.

As a complement of the *WIG* files, *BED* files store disjoint regions of the genome that result from filtering and transforming the signal data. Such filtering reduces the amount of data to comparably few disjoint regions with putative biological relevance.

In the context of this thesis, from here, the term *epigenomic data* means *preprocessed epigenomic data*.

2.3 Epigenomics mapping consortia

Due to technical advances described in Section 2.2, massive volumes of molecular data are now routinely generated by different epigenome mapping projects. However, even if a single project may have resources for mapping a subset of human cell types, due to the complexity of the human epigenome, more data and analysis efforts involving different epigenomes are necessary for understanding the complex connections in the healthy and diseased states.

The amount of resources to collect epigenomes for all tissues is enormous. Besides, a single epigenetic mark, e.g., DNA methylation is not enough to elucidate the existing questions about gene expression, cell development, and implications in health, therefore needing different experimental data and thus demanding more resources.

In this case, more data, samples and experimental data from different projects must be collected, aiming to build an epigenome atlas and to perform integrative analysis

⁸ Workflows are structurally more complicated than sequential pipelines

⁹ <https://genome.ucsc.edu/goldenpath/help/bigWig.html>

on this data; hence collaboration is the best approach for the task of deciphering the different epigenomes of the human cells.

To support this task, the *International Human Epigenome Consortium (IHEC)* (Stunnenberg *et al.* 2016) was established to promote cooperation between these different epigenome mapping projects. The principal goals of *IHEC* are to coordinate the production of reference maps of human epigenomes for key cellular states relevant to health and disease, to facilitate rapid distribution of the data to the research community, and to accelerate the translation of this knowledge for improving human health. A critical component of *IHEC* is to coordinate the development of common bioinformatics standards, data models and analytical tools to organize, integrate and display the epigenomic data generated (Stunnenberg *et al.* 2016). The cooperation between the *IHEC* members is structured into working groups on the following topics:

- Data standards defines the assays required, which epigenetic marks are mapped, and to define standardized protocols and *Quality Control (QC)* metrics for each experiment.
- Metadata standards defines the structure of the metadata for samples, experiments, and analysis results. The definition applies to the format and content of the metadata as well as which controlled vocabularies and ontologies must be used for describing the data semantically.
- Integrative analysis focuses on tools for data analyses and performing data analysis integrating the data from the different *IHEC* member projects’.
- Data Ecosystem promotes the sharing of the datasets produced by the *IHEC* members defining methods and standards for such task. Here the goal is to enable the reuse of epigenomic data in different purposes, e.g. integrative data browsing, visualization, and data analysis.
- Bioethics addresses the ethical foundation of epigenetic science, and provide advice on policy.
- Communication coordinates the communication activities between the committees and the working groups, as well as ensuring that *IHEC* activities and achievements are communicated effectively to the rest of the community and the wider public.

During the writing of this text, the *IHEC* member projects are:

- DEEP* funded by the *BMBF* from 2012 to 2017 is focused on the analysis of cells connected to complex diseases with high socio-economic impact, in their case: metabolic diseases as well as inflammatory diseases. It generates reference epigenomes for DNA methylation and histone modification maps as well as transcriptome data for white blood cells (lymphocytes, macrophages, and monocyte) cells, fat cells, and liver cells. (<http://www.deutsches-epigenom-programm.de>)
- BLUEPRINT* Epigenome Project funded by the European Union from 2011 to 2016 is focussed on hematopoietic cells from healthy individuals and their malignant leukemic counterparts. It generated DNA methylation, histone marks, and transcriptome data. The *BLUEPRINT* involves 42 European universities, research institutes, and industry partners from 12 countries. (Adams *et al.* 2012; Martens and Stunnenberg 2013, <http://www.blueprint-epigenome.eu>)

- Canadian Epigenetics Environment and Health Research Consortium (CEEHRC)* funded by the *CIHR* from 2011 to 2018 is composed of two epigenome mapping centers located in Vancouver and Montreal. It produces reference epigenomes for DNA methylation and histone modification, as well as transcriptome data. Whole genome sequences, small RNA, and other data types are also available for some samples. It produces reference epigenomes of the human blood, breast, brain, fat, kidney, muscle, skin, sperm, iPS cells, tonsils, intestine cells, and thyroid. (<http://www.epigenomes.ca>)
- REMC* funded by the *NIH* from 2008 to 2017 generates maps of DNA methylation, histone modifications, chromatin accessibility and small RNA transcripts in stem cells and primary ex vivo tissues selected to represent the normal counterparts of tissues and organ systems frequently involved in human disease. (Roadmap Epigenomics Consortium *et al.* 2015, <http://www.roadmapepigenomics.org/>)
- NHGRI ENCODE* funded by the *NIH* from 2003 and it is currently in its fourth phase, is the most comprehensive epigenome mapping project. The project generates transcriptomes, RNA binding, genotyping, DNA binding, DNA methylation, histone modification, and accessibility; and 3D chromatin structure maps of human tissues, cell types, and cell lines. (ENCODE Project Consortium 2012, <https://www.encodeproject.org/>)
- IHEC Team Japan* funded by *AMED* and *CREST* from 2011 to 2018 produces reference epigenomes for DNA methylation and histone modification maps as well as transcriptome data for the human gastrointestinal epithelial cells, vascular endothelial cells, and cells of reproductive organs. (<http://crest-ihec.jp>)
- Hong Kong Epigenomics Project* started in 2016 and it is still ongoing has the goal of generating reference epigenome maps of normal and diseased cells and making the data publicly available in order to accelerate scientific discoveries on the fields of neurobiology, aging, muscle biology, liver biology, and regenerative medicine. (<http://epihk.org>)
- KNIH Korea Epigenome Project* was launched in 2012 and it is still ongoing has the goal of producing 50 epigenomic datasets containing DNA methylation, ChIP-seq, and transcriptomes maps data of the pancreas, fat, kidney, muscle, and adipose tissue. (<http://152.99.75.168/KEP/>)
- Singapore Epigenome Project* established in the Genome Institute of Singapore is an ongoing project and aims to explore the epigenomes of human diseases starting with heart failure and autism spectrum disorder. It generates DNA methylation profiles, cardiac chromatin organization, and interaction between cardiac genes and their regulatory elements. (<http://ihec-epigenomes.org/about/ihec-countries/sg/>)

2.4 Other initiatives

Various independent groups are generating epigenomic data for helping them to answer their biological questions. For a research to be published, the data used in the manuscripts must be deposited in public repositories, like the *Gene Expression Omnibus*

(GEO)¹⁰, DNA Data Bank of Japan (DDBJ)¹¹, European Nucleotide Archive (ENA)¹², and ArrayExpress¹³.

In order to be usable by different research groups, epigenomic data must not only be deposited and accompanied by meaningful metadata, but the data must be processed using the same processing pipeline, including same parameters, software, and their respective versions¹⁴.

A vast variety of parametrizations of workflows comprising numerous tools is possible, hindering the comparison of data from data stored in the public repositories. For easing this problem, the projects Remap (Chèneby *et al.* 2017) and ChIP-Atlas (Oki *et al.* 2018b) are reprocessing the epigenomic data with uniform pipelines and enabling the free use of the processed data:

Remap Atlas reprocessed 485 datasets containing TFs, Transcription Coactivator (TCAs), and Chromatin-Remodeling Factors (CRFs). The processed datasets are available for browsing or download at its web page <http://tagc.univ-mrs.fr/remap/>.

ChIP-Atlas reprocessed the public ChIP-seq data submitted to the *Sequence Read Archives* (SRA): National Center for Biotechnology Information (NCBI), DDBJ, and ENA. It offers more than 69,000 experimental datasets reprocessed in an unified pipeline in their web page <http://chip-atlas.org>. It also provides web-tools for analyzing and exploring the data.

¹⁰ <https://www.ncbi.nlm.nih.gov/geo/>

¹¹ <https://www.ddbj.nig.ac.jp/index-e.html>

¹² <https://www.ebi.ac.uk/ena>

¹³ <https://www.ebi.ac.uk/arrayexpress/>

¹⁴ Ideally, even by the same wet lab protocols and tools, but this would not be feasible.



3

Accessing and analyzing epigenomic data

Generating epigenomic data is just the very first step for answering biological questions. It is followed by the data analysis process, where a researcher (i) searches in the data portals for the needed data; (ii) downloads the data files manually or using complex scripts; (iii) converts and filters the data into the desired format and content; (iv) analyzes the data using tools in their local computers, or analyzes the data directly in web applications. This analysis usually starts with data exploration followed, if necessary, by prediction methods and (v) the results are analyzed based on statistical methods.

This chapter presents methods and tools for accessing and analyzing epigenomic data. Section 3.1 presents the primary methods for accessing the public epigenomic data. Section 3.2 introduces the tools used for analyzing the epigenomic data locally, while Section 3.3 presents tools for exploring the data, focusing on online web tools. The chapter finishes with Section 3.4, which presents the tools and methods for performing predictions on the epigenomic data.

3.1 Accessing epigenomic data

Data portals are the most frequently used channel for distributing epigenomic data. They started as plain web pages listing the available data and links to the current data and, when available, to the metadata files. The *UCSC Table Browser Retrieval Tool* (Karolchik *et al.* 2004) was one of the first epigenomic data portals to be widely used. The *Table Browser* contains a simple web page where users can select and download the desired (epi)genomic datasets. It contains genomic annotations, like CpG islands, repetitive elements, as well as datasets from the initial *ENCODE* (ENCODE Project Consortium 2004) releases.

Although only a rudimentary way for finding, accessing, and retrieving data, epigenomic data portals are still the main epigenomic data distribution hubs. For example, the initial *ENCODE* data portal¹ was its main data distribution hub from 2007 to 2012, after being replaced by a new data portal² (Sloan *et al.* 2015), which supports full-text search, faceting searches using metadata segmentation, and an API that provides access to the files' metadata programmatically. Other projects, like *BLUEPRINT*³ and *ROADMAP*⁴, *DEEP*⁵, and *CEEHRC*^{6,7} also provide their own data portals.

¹ <http://genome.ucsc.edu/encode/downloads.html>

² <https://www.encodeproject.org/>

³ <http://dcc.blueprint-epigenome.eu/>

⁴ https://egg2.wustl.edu/roadmap/web_portal/

⁵ <http://deep.dkfz.de>

⁶ <http://www.epigenomes.ca/data-release/hg38/>

⁷ <http://epigenomesportal.ca/edcc/index.html>

The presented data portals offer *processed* epigenomic data with their metadata, but they do not have a common metadata format and the metadata files content is not fully integrated with their respective data files. For improving this situation, the *IHEC* data portal⁸ (Bujold *et al.* 2016) centralizes the data generated by its member projects as well as enforces a common metadata format⁹ with an API for listing and searching datasets based on their metadata.

Another effort, the *The Epigenome Reference Registry* (EpiRR) provide a repository of datasets metadata grouped by their BioSource and epigenetic marks, including links to the raw data (rather than to the processed data) in public sequence archives. But they are not easily analyzed, requiring a processing workflow to transform this data into data ready for data analysis.

Finally, these data portals provide terabytes of data. Generally, users are interested in only a fraction of them, but it is not possible to work on this data without downloading whole files, for local data filtering, and then, analysis.

3.2 Data analysis tools

After downloading the epigenomic data, it is necessary to filter for content of interest and also format the data to the desired format. The filtering and formatting can be performed using scripts, usually developed in scripting programming languages such as *R*, *Perl*, and *Python*. In combination with these scripts, command line tools for filtering and combining epigenomic data such as *BED Tools* (Quinlan and Hall 2010; Quinlan 2014), *BEDOPS* (Neph *et al.* 2012), *Tabix* (H. Li 2011), and *WiggleTools* (Zerbino *et al.* 2014) are widely used.

Software libraries are being developed for empowering the development of bioinformatics tools and data analysis scripts. Usually, these libraries are organized under an umbrella project. For example, there is *Bioconductor* (Gentleman, V. J. Carey, Bates, Bolstad, Dettling, Dudoit, Ellis, Gautier, Ge, Gentry, *et al.* 2004a) for the *R* programming language and *BioPython* (Cock *et al.* 2009) for *Python*. *Bioconductor* and *BioPython* are developed and tailored specific to bioinformatics problems, such as loading, visualizing, analyzing, and performing predictions on the data. Although they are a major help in epigenomic data analysis, command line tools and libraries require that users have programming proficiency. Hence, the average biomedical researchers cannot make use of them.

Besides programming and command line tools, which require experienced bioinformaticians to operate them, tools like *Taverna* (Wolstencroft *et al.* 2013) and *Galaxy* (Gardine, Riemer, Hardison, Burhans, Elnitski, Shah, Y. Zhang, Blankenberg, Albert, Taylor, *et al.* 2005b) provide a visual interface, where researchers can develop their data workflow, from acquiring the data, processing, and analyzing it in a visual and programming-free environment.

⁸ <http://epigenomesportal.ca/ihec/>

⁹ https://github.com/IHEC/ihec-metadata/blob/master/specs/Ihec_metadata_specification.md

In short, the data analysis can be performed by (i) developed scripts, using existing libraries and command line tools, or (ii) using visual data workflows. These two methods usually require the download of massive datasets for selecting only a fraction. Furthermore, due to the data size, its manipulation is not feasible on a local workstation, requiring more powerful computational environments, not available to all researchers. Existing tools need data in specific formats, such that usually users must convert their data for fitting the tools' requirements.

Finally, many of these tools do not scale with the growing amount of epigenomic data. They require vertical computational scaling, which means that each computational node needs to scale with the data volume. It results in the acquisition of more powerful, and consequently, more expensive computational resources. An opposite approach, the horizontal scaling is based on increasing the number of available computer resources, where the computational power of the nodes stays constant and the number of nodes scales. Consequently, many of the epigenomic data analysis tools are not ready for cloud computing, where the computational resources scale horizontally.

3.3 Data exploration tools

Data exploration is the first analysis to be applied to new datasets or with new questions. This process uses visual elements to facilitate the interpretation of the data and guide the researcher through its understanding. Presently, epigenomic data can be explored using genome browsers, local tools, and web tools. In the following sections, these three general techniques are presented together with their main tools.

3.3.1 Genome Browsers

Genome browsers (Stein *et al.* 2002) were the first tools for visually exploring epigenomic data. Genome browsers provide a graphical interface for users to browse, search, retrieve and analyze the genomic sequence and annotation data (J. Wang *et al.* 2012). Figure 3.1 shows the UCSC *Genome Browser* displaying information around the gene *HOXA1* with some epigenetic marks.

Different genome browsers are in widespread use. The two most frequently used genome browsers are UCSC *Genome Browser* (Kent *et al.* 2002) and *Ensembl Genome Browser* (Hubbard *et al.* 2002; Stalker *et al.* 2004; X. M. Fernández and Birney 2010). Genome browsers can also be embedded in other projects, such as *GBrowser* (Donlin 2009) and *JBrowser* (Skinner *et al.* 2009).

Even though *Genome browsers* are widely used, they rely strongly on the experience and knowledge of the researcher, and only a small fraction of the available data can be visualized simultaneously. Genome browsers are not scalable because they cannot cope with the growing amount of epigenomic data, not only in the sense of storing and manipulating the existing (epi)genomic data, but also for visualizing, comparing, and analyzing these data.

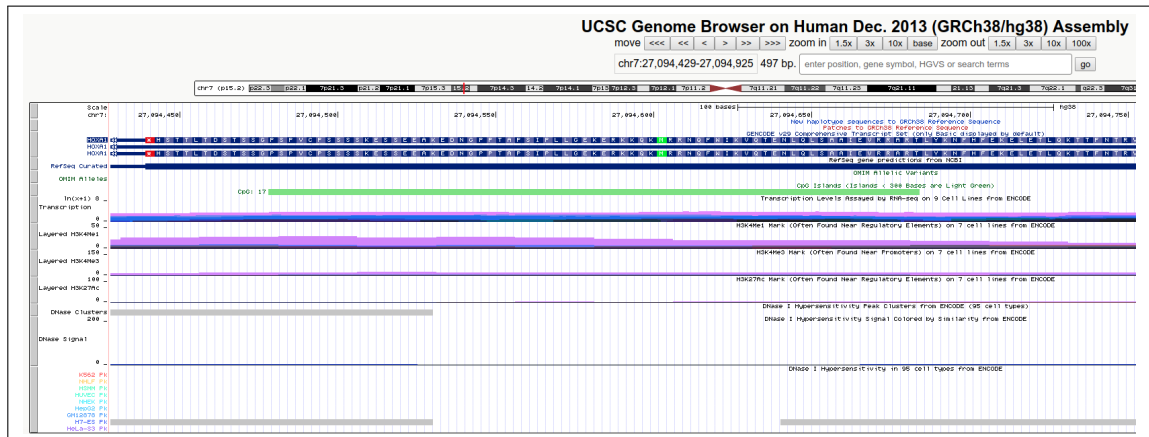


Figure 3.1: UCSC Genome Browser displaying information around the gene *HOXA1*. It displays the gene body, transcription region, histones marks (*H3K4Me1*, *H3K4mM3*, *H3K27Ac*), and DNaseI hypersensitive site regions.

3.3.2 Local analysis tools developed in the framework of DEEP

Many tools have been developed for analyzing epigenomic data locally. Here, two tools developed in the context of the *DEEP* project with the goal of providing automatic reports on the epigenomic data are presented: *deepTools*¹⁰ (Ramírez *et al.* 2014; Ramírez *et al.* 2016) and *RnBeads*¹¹ (Assenov *et al.* 2014).

deepTools

The software suite *deepTools* is a set of tools developed for analyzing *ChIP-seq*, *RNA-seq* or MNase-seq data. This package provides command line tools and an API that may be used to integrate the tools into a Galaxy Workflow. The components of *deepTools* handle *raw* data files (*FASTQ* and *BAM*) as well as *processed data* (*BED* and *WIG*), and can (i) calculate the correlation between different data files, which may represent different epigenetic marks or BioSources; (ii) visualize the genome coverage of a data file; (iii) check the CG bias; (iv) assess the strength of *ChIP-seq* signal data. Figure 3.2 presents the analysis workflow that can be performed by *deepTools* and its steps: data quality analysis and downstream analysis.

RnBeads

RnBeads is a package for comprehensive analysis of DNA methylation data. The tool implements an analysis workflow with functionalities that go beyond the previously existing tools. *RnBeads* has two main advantages: (i) it is user-friendly, in that the users must only inform where the data is and start the workflow; (ii) the analysis result document is a self-contained readable hypertext report; (iii) it scales to large sample sizes. The tool also offers state-of-the-art normalization techniques, experimental quality control, CpG and sample filtering, batch effect, phenotype identification, and analysis characterization of differential methylation between groups of samples. Figure 3.3 presents

¹⁰ <https://deeptools.readthedocs.io>

¹¹ <https://rnbeads.org/>

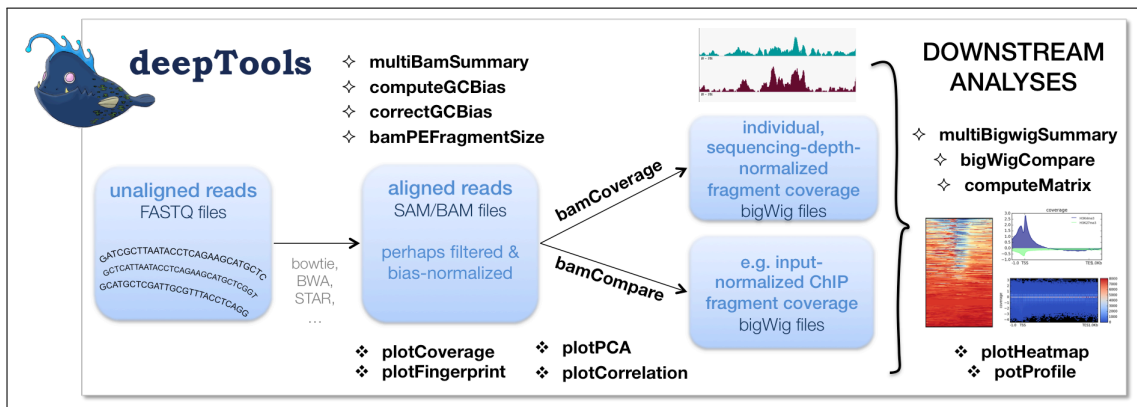


Figure 3.2: The *deepTools* analysis workflow analyses the data and generates plots for the data quality, correlation between different data files, and heatmaps and profiles for downstream analysis. (<https://deeptools.readthedocs.io>).

the *RnBeads* workflow: from the input files from different formats and content, the pre-processing and quality control process, and the analysis steps, finishing with the reports and output data files.

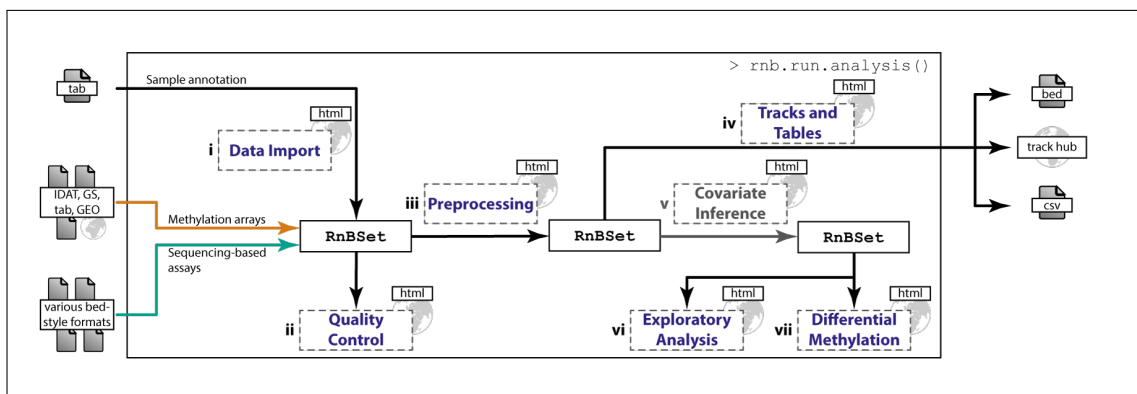


Figure 3.3: RnBeads workflow: the input data can have different sources and formats. The tool affords preprocessing, generating quality controls and preprocessing reports, followed by another set of analysis, producing reports on covariance, also for exploratory analyses and differential methylation, and creating data files that can be used in other data analysis tools (<https://rnbeads.org/>).

3.3.3 Web analysis tools in the context of DEEP and BLUEPRINT

The previously presented tools perform analysis using local infrastructure, which comes with inconveniences, for example, the time to set up the software and the need of having all the data locally. Besides, it is hard to keep in pace with the growing amount of available epigenomic data, for which it is necessary to build a computational and software infrastructure for storing and processing this data.

Adding to the technical issues, the local tools are usually not suitable for analyzing and comparing hundreds or even thousands of data files simultaneously. Their analysis and visualization methods do not scale for such an amount of data.

Here, two data exploration web tools, developed in the context of *DEEP* and *BLUEPRINT*, are presented: The *BLUEPRINT* Data Analysis Portal¹² (J. M. Fernández *et al.* 2016) and *EpiExplorer*¹³ (Halachev *et al.* 2012).

BLUEPRINT Data Analysis Portal

The *BLUEPRINT* Data Analysis Portal provides an online interface for the comparative analysis of epigenomes of hematopoietic cell types generated by the *BLUEPRINT* project. Its interface affords an interactive exploration of genomic regions, genes, and pathways. Figure 3.4 shows the start screen, where users can type (epi)genomic terms and genomic locations for exploring its relationship with other (epi)genomic elements.

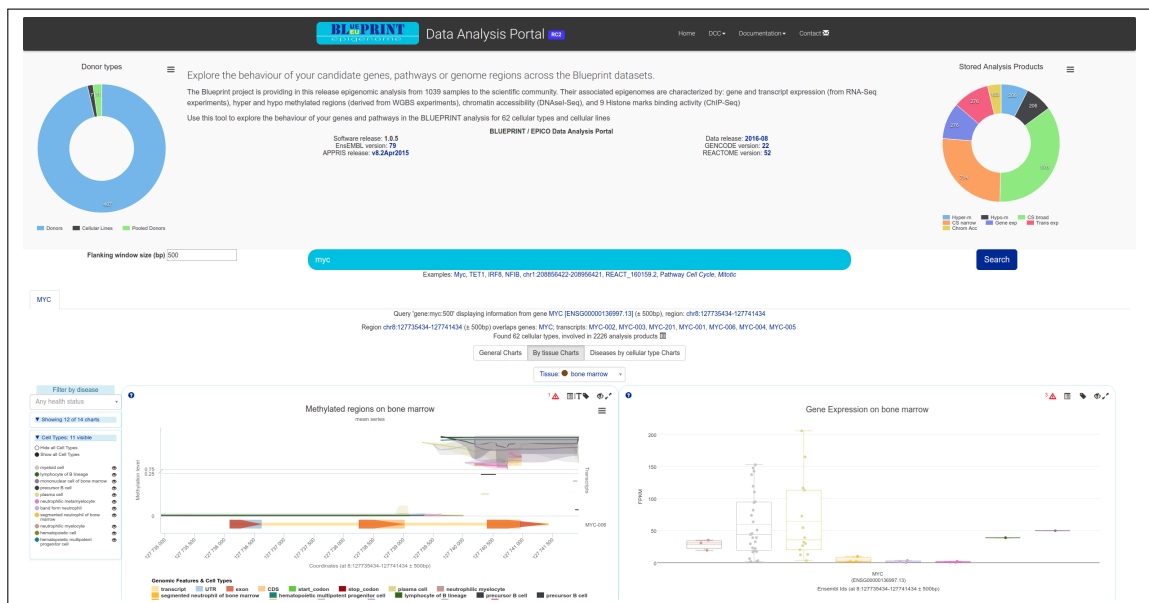


Figure 3.4: The *BLUEPRINT* Data Analysis Portal provides a web interface where users can type (epi)genomic terms and genomic locations for exploring its relationship with other (epi)genomic elements (<http://blueprint-data.bsc.es>).

EpiExplorer

EpiExplorer is a web data analysis tool that enables researchers to explore datasets using large reference epigenome datasets without complex scripting or laborious pre-processing. *EpiExplorer* provides an intuitive interface, where users can perform programming-free analysis, only clicking on web interface visual elements. Furthermore, due to the data preprocessing and indexing scheme, analyses are performed dynamically within seconds.

EpiExplorer brought a paradigm shift to the field of epigenomic data analysis: rather than focusing on a single genomic location, such as a gene, it provides a global analysis of the entire (epi)genomic space. This means that researchers have an overview of all

¹² <http://blueprint-data.bsc.es>

¹³ <https://epiexplorer.mpi-inf.mpg.de>

epigenetic marks and genomic elements in the entire genome, making it possible to analyse the relationship between different elements on a global scale.

Figure 3.5 shows the initial web page of EpiExplorer, displaying the comparison of two datasets, CpG Islands and 5hmC modifications, in relation to other (epi)genomic elements.

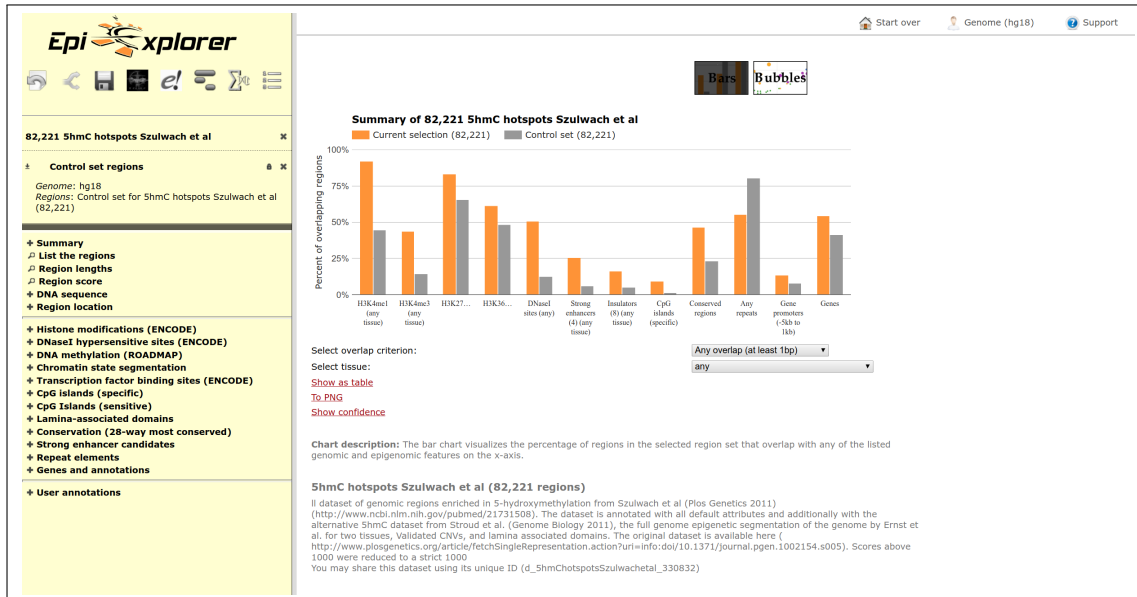


Figure 3.5: EpiExplorer’s initial web page, showing the comparison of two datasets, CpG Islands and 5hmC modifications, in relation to other (epi)genomic elements (<https://epiexplorer.mpi-inf.mpg.de>).

These tools provide ways for analyzing epigenomic data in a programming-free environment, but they work on pre-processed datasets. New epigenomic questions require analyzing of many different region-sets, usually not available in such tools. Furthermore, they are not ready for the increasing amount of data, since their visualization methods do not cope with large datasets, and they require initial pre-processing of the data for analysis, which is not feasible when considering thousands of experiments.

3.4 Predictive analysis tools

Predictive analytics is a set of statistical techniques from data mining, modelling, and machine learning, which analyze the data to make predictions about unknown events. In the epigenomic context, they are used for making predictions about the epigenomic data, for example, predicting DNA methylation levels, gene expression, and finding and quantifying the similarity between epigenomic datasets.

EpiGRAPH (Bock *et al.* 2009) was one of the first tools for performing prediction analysis on (epi)genomic data. *EpiGRAPH* applies machine learning to (epi)genetic datasets for the purpose of identifying sequence regions with similar characteristics. As usage example: (i) predict DNA methylation based on the DNA sequence and structure; (ii) discover the characteristics of Polycomb Response elements in mammals when compared to *Drosophila* using *ChIP-seq* data. In general, *EpiGRAPH* is a powerful tool, performing

many types of statistical analyses by applying machine learning to the epigenomic data. But the tool is complex, requiring detailed knowledge of machine learning on the side of the user. It also requires a complicated set up for performing the data analysis and prediction.

In the direction of facilitating the epigenomic data analysis, *LOLA* (Sheffield and Bock 2016) is an *R/Bioconductor* package that provides methods for analyzing genomic region overlaps between public and custom datasets. The tool compares genomic region files (*BED* files) with the regions of interest against a database of other datasets (from different studies and projects) to test for enrichment in their overlaps. The user can relate newly generated data and public datasets, create new hypotheses and annotate the new datasets. The *LOLA* method is described in detail in Section 4.5.7.

Enrichment analysis is used to investigate if there are more elements with certain properties than expected by chance, or if one set is overrepresented in a larger collection. One well-known example in bioinformatics is the gene-set over-representation analysis that assesses if a set of genes, for example, from a pathway, is overrepresented in a set of differentially expressed genes. *Locus Overlap Enrichment Analysis* (*LOLA*) expands this idea moving from genes to genomic regions.

As *LOLA* is an *R* package, it is necessary to have some knowledge in this programming environment. Besides, users must download and convert the required datasets to the format required by *LOLA*. *LOLA* results are stored in an *R* data structure, and thus not easy to visualize. For handling such issues, *EpiAnnotator* (Pageaud *et al.* 2018) was developed as a web interface, developed in *R/Shiny* that enables researchers to use the *LOLA* method in an intuitive interface.

Similarly, to *LOLA*, *GIGGLE* (Layer *et al.* 2018) is an (epi)genomic search engine that finds and ranks genomic locations shared by the query regions and regions presented in the database. It claims to scale to billions of genomic regions and is faster than existing methods. Similar to *LOLA*, it is possible to use the data from public epigenomic mapping projects to build the user's database, facilitating the data integration and hypothesis generation. In the same way as *LOLA*, it is required that users find, download, and convert the data to the format expected by *GIGGLE*.

Obtaining a biological insight from the epigenomic data is a complex task. Hence the boundaries between epigenomic data exploration and predictions become uncertain. In this direction, tools like *GenometriCorr* (Favorov *et al.* 2012) and *daVIE* (Fejes *et al.* 2014) empower users to explore the epigenomic data with the support of statistical methods. Another tool, *eFORGE* (Breeze *et al.* 2016) provides a web interface for analysis and interpretation of *Epigenome-Wide Association Study* (*EWAS*) data. Its input is a list of *EWAS* array probes for performing overlap enrichment analysis using 454 DNaseI hypersensitive site samples, helping to identify disease-relevant genomic regions in different cell types for several common diseases.

These tools strongly rely on local non scalable processing. Furthermore, there is no “one size fits all” tool in bioinformatics. Different problems need a different set of tools, and two important aspects are the quality of the tools and the communication and interchange of data between them. In this direction, *R/Bioconductor* and *Python* with statistical and machine learning packages provide powerful environments for analyzing and performing data prediction on the (epi)genomic data while visual and web tools provides a first overview and hypothesis generation on the users' data.

3.5 Requirements for large-scale epigenomic data analysis tools

The generated epigenomic data have the potential of unraveling the complexity of epigenomic regulation. However, the lack of simple access and analysis mechanisms hinder the practical use of this data. This chapter elaborates on open challenges in the large-scale epigenomic data analysis.

First, users must manually search the experimental metadata in different data portals to identify the ones that are valuable for their analysis. Therefore, it is imperative to **provide a unified searching tool for the data and metadata generated by the IHEC member projects**.

Second, the data and metadata of the experiments are not connected, making the identification of the ideal datasets a cumbersome task. Hence, it is necessary to **integrate the epigenomic data and metadata**.

Third, users need to perform manually or to develop one-time-use scripts for downloading gigabytes of data spawned in hundreds of files, only to extract the regions of interest corresponding to a fraction of the entire downloaded data. Moreover, it may be necessary to manipulate the data across many files, being essential to **provide methods for filtering and transforming the data before being downloaded**.

Fourth, many epigenomic data analysis tools require that the data is locally available. For this purpose, the data must be transformed, but complex operations on these data are not always feasible on a local computer due to resource limitations, making it necessary to **provide means of investigating the epigenomic data in environments with limited resources**.

Fifth, the use of cloud computing is growing and becoming standard for dealing with different types of data. Large-scale epigenomic analyses may benefit from such infrastructure, but it is necessary to **develop tools that are scalable and cloud-ready**.

Sixth, connecting data from different formats and sources is an unwieldy task, causing researchers to spend a considerable amount of time and resources. Therefore, it is imperative to **handle and to provide (epi)genomic data in configurable formats, and to deal with heterogeneous data**.

Seventh, current technological advances shifted the gene-centric studies to trans-genomic or genome-wide analysis. Consequently, new tools must **access, manipulate, and analyze data dynamically from regions of the whole genome**.

Eight, many epigenomic data analysis processes involves the development of numerous software (*scripts*), and for this, learning at least one programming language is required. The user also needs to be able to apply data analysis methods on the data, whereby different programming languages provide different forms of performing these tasks. For facilitating the scripts development, it is necessary to **provide a comprehensive set of operations accessible from different programming languages**.

Ninth, epigenomic data analysis is a complex task particularly for researchers without programming skills. For facilitating the use of the epigenomic data for researchers without programming skills, it is fundamental to **provide tools for analyzing epigenomic data in a programming-free environment and user-friendly interface**.

Finally, tenth, new technologies are producing more data than researchers can analyze. Tools that **cope with the amount of generated epigenomic data, not only in finding and manipulating but also for visualizing and analyzing it** are necessary.

The *DeepBlue Epigenomic Data Server* and its ecosystem aim to fulfill these requirements and represent a robust and comprehensive set of tools for large-scale epigenomic data analysis. The *DeepBlue Epigenomic Data Server* provides a set of tools which each one has the goal of providing one or more of the previously listed requirements. This thesis presents the entire *DeepBlue Epigenomic Data Server* ecosystem in the following chapters, showing how the *DeepBlue* tools and methods fulfill these requirements.

4

DeepBlue Epigenomic Data Server



This chapter describes the DeepBlue Epigenomic Data Server and its components. They were implemented by the author with support of Fabian Reinartz and Natalie Wirth. The usage example in the Section 4.6.4 was performed with support of Dr. Masanori Honsho. The DeepBlue Epigenomic Data Server was published in [Albrecht et al. 2016](#).

Epigenome mapping projects are generating large volumes of epigenomic data with the promise of improving the understanding of cell regulation. Obtaining and organizing this data is a cumbersome task. Currently, epigenomic data is only accessible through web portals that offer access to experimental data files and their metadata but these web portals still lack adequate mechanisms for searching and obtaining these data programmatically, it means through a software code, rather than through a user interface. Moreover, the absence of metadata standardization complicates the use and integration of data from different sources in a single study. Finally, researchers must process and analyze terabytes of epigenomic data, which is not effective on local computers.

The *DeepBlue Epigenomic Data Server* was developed to support researchers in finding, handling, and obtaining the necessary (epi)genomic data for their analysis. The *DeepBlue Epigenomic Data Server* solves the issues of searching, manipulating, and downloading the epigenomic data by providing: (i) a collection of (epi)genomic experiments and annotations files with their metadata; (ii) means for finding and selecting the appropriate epigenomic data; (iii) data operations in the form of a comprehensive API, where

users can find, select, filter, manipulate, enrich, and retrieve the relevant (epi)genomic data programmatically; (iv) an online server that processes the epigenomic data directly.

This chapter presents the *DeepBlue Epigenomic Data Server*, and it is structured as follows: Section 4.1 introduces the reader to the *DeepBlue Epigenomic Data Server* and its ecosystem. Section 4.2 presents the *DeepBlue Server*, the central component of the *DeepBlue Epigenomic Data Server* ecosystem. Section 4.3 explains how the data is stored into and retrieved from the *DeepBlue Server*. Section 4.4 introduces the populator, the tool used to import the (epi)genomic data from different sources. Section 4.5 presents the *DeepBlue Server* API. Section 4.6 demonstrates the *DeepBlue Server* functionalities through a set of use cases. Section 4.7 discusses the capabilities and usage of the *DeepBlue Epigenomic Data Server*. Appendix A.2 contains implementation details.

4.1 Overview

Epigenomic mapping consortia are generating thousands of epigenomic data files, each containing kilobytes to gigabytes of data, totalling dozens of terabytes of epigenomic data ready to be analyzed and used. Usually, researchers need only a small fraction of the available epigenomic data to answer their biological questions. Using the currently available data repositories to obtain the necessary data requires users to download several files amounting to gigabytes of data that afterwards need to be handled locally, e.g., filtered, merged, transformed, and aggregated. Storing such an amount of data and applying data transformation on it usually requires extensive computational resources, which are not regularly found on local computers.

The *DeepBlue Epigenomic Data Server* was developed to facilitate access to and analysis of publicly available epigenomic datasets. Its main component, the *DeepBlue Server* has as primary responsibilities: (i) storing and organizing the (epi)genomic data and (ii) making this data readily accessible for researchers.

These responsibilities are fulfilled by the *DeepBlue Server* by providing a collection of experiment and annotation files with their metadata information. The server also provides a comprehensive API which supports users in listing, searching, selecting and operating on, the data programmatically directly on the server, downloading only the results containing the necessary data.

The API also enables users to perform complex data operations, such as filtering, summarizing, and performing enrichment analyzes on the (epi)genomic data. The API operations can be combined into custom workflows, thus offering nearly the same degree of flexibility as a local programming environment. Therefore, the *DeepBlue Server* merges the facilities of accessing the epigenomic data from different projects in a single location with a comprehensive API.

Access to the *DeepBlue Server* is facilitated by XML-RPC and *RESTful* protocols, both of which are web standards. These protocols are supported by major programming languages, making *DeepBlue* a language-agnostic system. However, the *DeepBlue Epigenomic Data Server* ecosystem also supports users that have little or no expertise in programming with a user-friendly web portal for browsing and downloading the available data (Chapter 6) and a powerful data analysis web tool (Chapter 7).

The *DeepBlue Server* keeps track of all user operations and assigns query *IDs* to each step of an analysis. These *IDs* can be used flexibly between different API endpoints.

This means that experiment files of interest to the user can be selected in the web portal and subsequently be used in other platforms such as in programming scripts to retrieve the data for follow-up analysis. A complete use case is presented in Section 6.3.1.

The *DeepBlue Server* provides access to more than 60,000 experimental epigenomic files from the major epigenome projects: *AMED-CREST*, *CEEHRC*, *DEEP*, *BLUEPRINT* Epigenome Project, *REMC*, *ENCODE*, and *ChIP-Atlas*. The *DeepBlue Server* was developed following established *IHEC* standards and the *DeepBlue Server* development team engaged in frequent exchange with *IHEC* members for including additional data. *DeepBlue Server* also provides gene models, annotations such as CpG island, DNA sequences, and ontologies. A list of the imported data is available in Appendix A.1.

The *DeepBlue Server* contains an extensive documentation, including a user manual¹, coding examples², use cases³ and *Jupyter* notebooks⁴, and a complete API⁵ reference. While the online coding examples are implemented in *Python*, the *R/Bioconductor* package (Chapter 5) provides documentation and examples in *R* programming language.

4.1.1 DeepBlue Epigenomic Data Server ecosystem

The *DeepBlue Epigenomic Data Server* and its ecosystem is composed of six components: (i) the data server, named *DeepBlue Server*; (ii) the data importer, named *Populator* (Section 4.4); (iii) the Web Middleware (Section A.2.5), (iv) the Web Portal (Section 6.2), (v) the Web Epigenomic Data Exploration Tool: *DIVE* (Chapter 7); and (vi) the *Bioconductor/R* package (Section 5). Figure 4.1 shows all these components and how they are connected to each other. Each component is described individually in the following sections and chapters.

4.2 DeepBlue Server

The *DeepBlue Server* is the central component of the *DeepBlue Epigenomic Data Server* ecosystem. It manages and controls data access, handles user operations, and processes the requests. Section A.2 contains implementation details. The *DeepBlue Server* implements more than 250 software integration tests⁶. Each integration test executes a set of *DeepBlue Server* API operations and the operation results are compared to expected result, that are previously obtained by manually executing the same tasks. These tests are implemented in *Python*. All *DeepBlue Server* API operations have at least one integration test associated with it. The use of tests improve the *DeepBlue Server* software quality by reducing the chance of development flaws being introduced during the software development.

¹ <https://deepblue.mpi-inf.mpg.de/manual/>

² <https://deepblue.mpi-inf.mpg.de/examples.php>

³ https://deepblue.mpi-inf.mpg.de/use_cases.php

⁴ <https://deepblue.mpi-inf.mpg.de/notebooks>

⁵ <https://deepblue.mpi-inf.mpg.de/api.php>

⁶ Software integration test is a type of testing where software modules are combined and tested as a group.

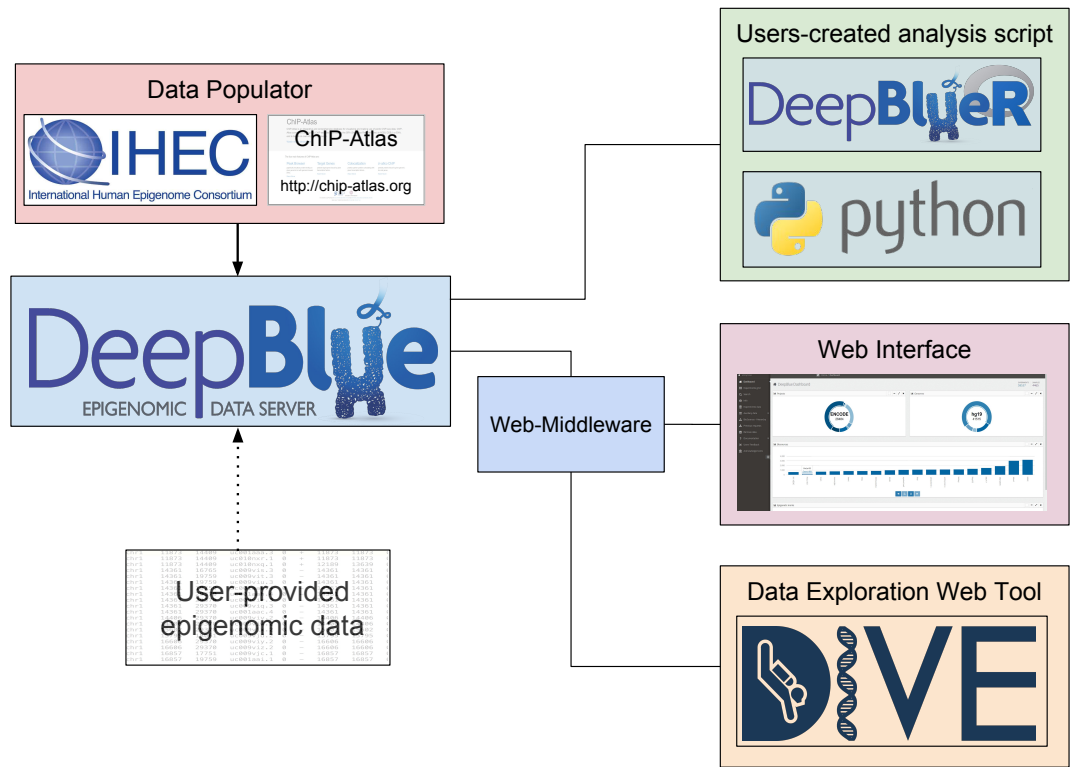


Figure 4.1: DeepBlue Epigenomic Data Server ecosystem overview: this diagram shows the *DeepBlue Server* as the central piece of the *DeepBlue Epigenomic Data Server* ecosystem. The *Populator* inserts the data from the epigenomic portals, and the server makes it available to the users via programming scripts (*R* and *Python*), Web Portal, and *DIVE*.

4.2.1 DeepBlue Server Components

The *DeepBlue Server* is composed of three main components: (i) an *XML-RPC Server* that receives and handles user requests; (ii) a *Processing Engine* that performs operations on the data; (iii) a *Database Access* point that is connected to a MongoDB database instance and is responsible for storing and retrieving data. Figure 4.2 shows the *DeepBlue Server* architecture and its sub-components, which are detailed in the following sections.

XML-RPC Server

The *DeepBlue Server* is accessed using the XML-RPC protocol. It is a *Remote Procedure Call (RPC)* protocol which uses *XML* to serialize and the *HTTP* protocol to transport the operation calls and answers between users and the *DeepBlue Server*.

The XML-RPC protocol was chosen because it is simple to use and transparent, where XML-RPC operation calls appear as local operation calls for the users. Besides, it is simple to implement a server compatible its specification. But the most important feature of this protocol is that virtually all main programming languages support it: access and

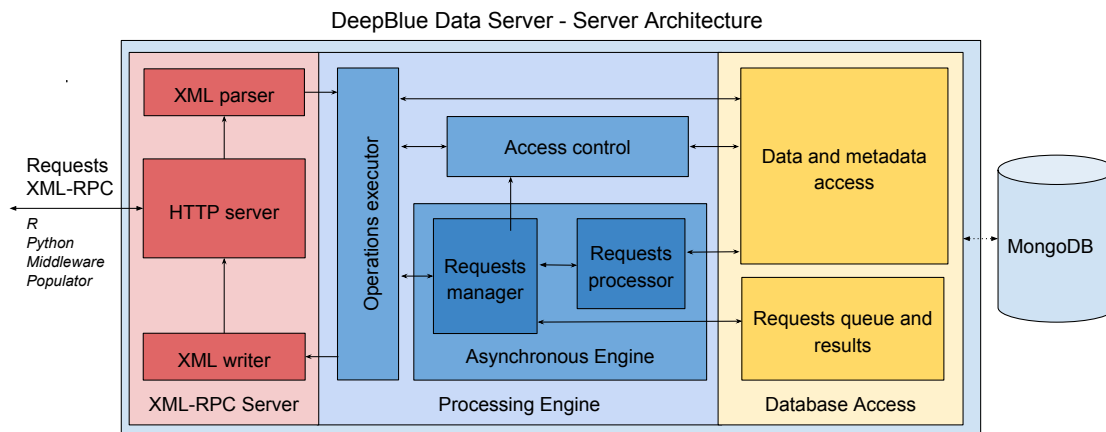


Figure 4.2: DeepBlue Server Architecture: the *DeepBlue Server* is composed of: (i) an XML-RPC server that receives and handles user operations. It is composed of the *HTTP* server that provides the communication with the user. As the communication is made through the XML-RPC protocol, the *XML-RPC Server* also contains a *XML* parser and writer; (ii) a *Processing Engine* that performs operations on the data. This component is responsible for managing and executing the requests, as well as perform the data and operations access control; (iii) a *Database Access* component that stores and access the data using the MongoDB database.

usage of *DeepBlue Server* was performed in *Python 2*, *Python 3*, *JavaScript*, *TypeScript*, *PHP*, *R*, and *Java* programming languages.

The *DeepBlue Server* embedded *XML-RPC Server* is implemented using Boost ASIO⁷ library for providing the *HTTP* server and the Expat library⁸ for parsing the *XML* content. The *HTTP* server implements a subset of the *HTTP* protocol, accepting solely the *HTTP POST* method.

The *XML-RPC Server* receives requests containing the operations to be executed using the *HTTP* protocol. The *XML* parser extracts the request's *XML* content. The content is converted to an internal representation that is executed by the *Processing Engine* (Section 4.2.1). After the request execution, the result is returned to the *XML-RPC Server*, where the *XML writer* serializes the result to a *XML* representation compatible with the XML-RPC protocol, and this content is returned to the user using the *HTTP* protocol.

Processing Engine

The *Processing Engine* executes all *DeepBlue Server* operations. As the first execution step, it uses the *Access Control* component for verifying whether the user has permission for performing the requested operation. If this is not the case, an error message is returned to the user. *DeepBlue Server* operations can be divided into two main execution categories: synchronous and asynchronous. Synchronous operations are processed, and

⁷ https://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio.html

⁸ <https://libexpat.github.io/>

the result is returned immediately to the user after completion. Usually, these operations do not require complex processing and their execution time never exceeds a few seconds. In contrast, asynchronous operations are more complex and are executed using the *Asynchronous Engine*.

Synchronous operations are executed directly by the *Operations executor*. Examples of synchronous operations are the *list_experiments* and *info* operations that are presented in Section 4.5. The *Operations executor* uses the *Database Access* component for accessing the data and processing the requests.

The *Asynchronous Engine* uses the *Request manager* for queueing and executing operation requests: new requests are included in a queue, named *jobs queue*, and a request identifier is returned to the user. This identifier is used for obtaining the status of the request and for downloading the processed result. The *Request manager* is also responsible for storing the results from the *Request processor* in the database and for removing old requests.

The *Request processor* is responsible for processing all DeepBlue asynchronous requests. It retrieves the operations enqueued by the *Request manager* from the *jobs queue*, processes them, and delivers the result to the *Request manager* that stores the result using the *Database Access*. The *Request Processor* can execute multiple requests simultaneously, where the number of simultaneous requests is configured at startup (Section A.2.1). The asynchronous processing results are stored and can be accessed instantly after the processing is finished. The time limit and maximum memory consumption are also configurable.

Database Access

The *Database Access* component facilitates storing and retrieving data from the database. It facilitates storing and retrieving (epi)genomic data, metadata, user information, and operation requests including their results. The *Database Access* uses a MongoDB database instance for data storage. Section 4.3 details how this component stores the data and uses the MongoDB database.

4.2.2 Scalability

Due to the massive increase in available epigenomic data, scalability is an important issue. Existing tools must be prepared to handle massive growth in the amount of data. Acknowledging this issue, the *DeepBlue Server* was developed to cope with the increasing amount of available (epi)genetic data, being scalable in two respects: data storage and data processing. The data storage scalability relies on the MongoDB scalability facilities (Section 4.3), and the data processing scalability depends upon the existence of multiple independent processing nodes.

In the *DeepBlue Epigenomic Data Server* ecosystem, a processing node is an executing instance of the *DeepBlue Server*. It is required to have at least one executing *DeepBlue Server* instance, but several instances can be executed on the same or a different computer, connected to the same MongoDB instance. When using more than one *DeepBlue Server* instance, the additional instances can have their XML-RPC servers deactivated in order to act as a unique processing node. The processing nodes access the *jobs queue*, where they obtain the requests, process them, and store the result back to the MongoDB

instance. If a *DeepBlue Server* instance is receiving too many processing requests, more processing nodes can be activated in different computers, distributing its computation load.

Figure 4.3 shows an overview of a *DeepBlue Server* configured as a processing node. It is important to note the absence of the XML-RPC server. Furthermore, the data and metadata are accessed only for reading. In this example, the processing node obtains the processing requests from *Requests queue and results* component. This component uses the *jobs queue* stored in the MongoDB instance. After obtaining a processing request, the processing node handles it, and saves the result to the database using again the *Requests queue and results* component.

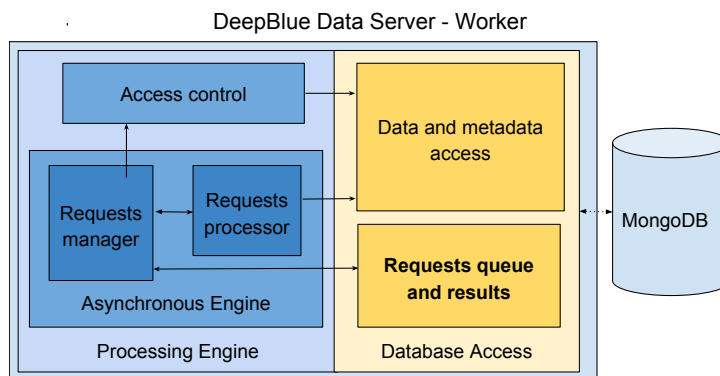


Figure 4.3: DeepBlue Processing Node: this *DeepBlue Server* instance is configured as processing node, without an XML-RPC server. It is solely responsible for handling the requests. It uses the *Requests queue and results* components for obtaining the requests stored in the *job queue*, which are stored in the MongoDB database instance.

As shown, the *DeepBlue Server* has a scalable architecture, in which multiple *DeepBlue Server* instances can be executed on different hardware servers, working as a computational cluster, for improving the processing of the requests stored in the *jobs queue*. A typical *DeepBlue Server* cluster arrangement has a master instance that receives the XML-RPC requests, processes the synchronous requests, and enqueues the asynchronous requests. Other *DeepBlue Server* instances are responsible for accessing the *job queue* and processing its requests.

Figure 4.4 shows an example with three different hardware servers: *DeepBlue Server* instance 1, 2, and 3. Instance 1 has a *DeepBlue Server* with the XML-RPC server enabled, but the *Requests Processor* is disabled. Instance 1 receives the processing requests from the users and enqueues them to the *jobs queue* using the *Processing manager*. Simultaneously, instances 2 and 3 access the *jobs queue*, each one retrieves a different processing request, perform the necessary computation, and store the result using the *Requests queue and results* component. This configuration also ensures that the XML-RPC server instances remain responsive even under high workload, because it is possible to include new *Processing Node* instances in additional hardware to increase the cluster throughput, and thus, minimizing the waiting time of the enqueued processing requests.

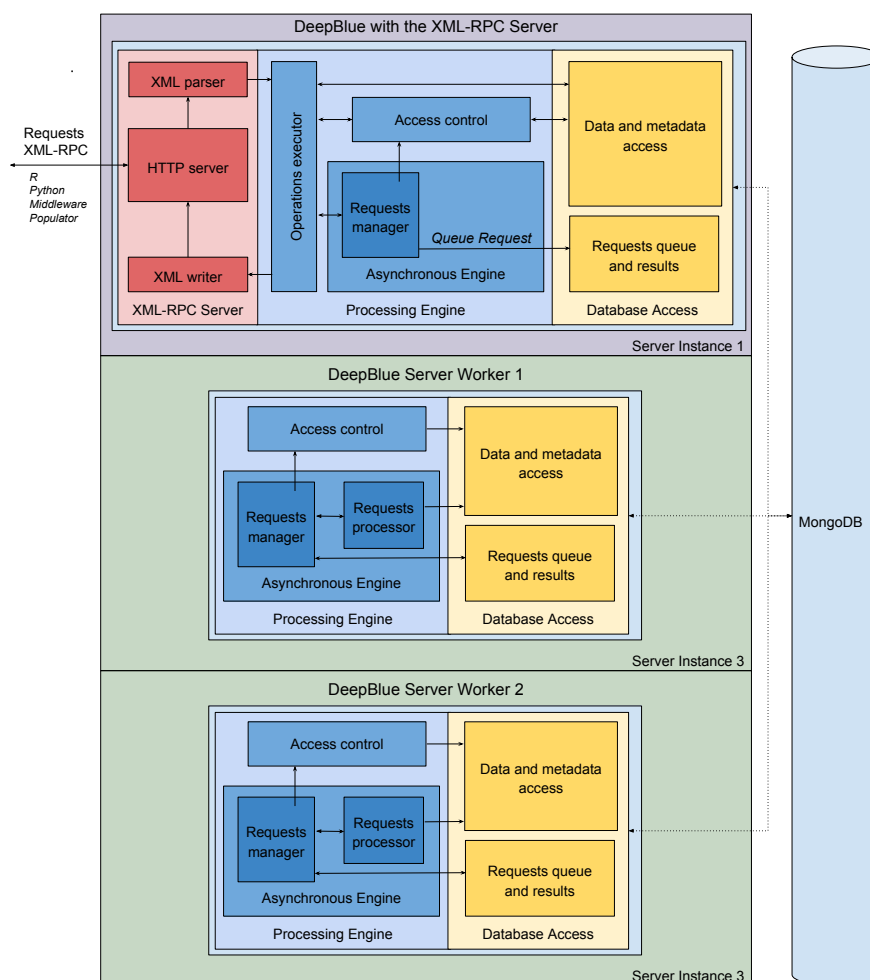


Figure 4.4: DeepBlue Server Cluster Example: three *DeepBlue Servers* working simultaneously, where the *DeepBlue Server* instance 1 is responsible to answer the XML-RPC requests and the servers 2 and 3 are responsible for processing the requests stored in the *jobs queue*.

4.2.3 Access control

Access control is facilitated with a *user_key* that is sent to the *DeepBlue Server* as a parameter in almost all operations. Users can obtain the *user_key* by registration or use an anonymous user key (*anonymous_key*).

Registered users have the benefit of storing private data, accessing previous processing analysis, and requesting more resources for executing *DeepBlue Server* API operations, for example, more processing time and memory.

The anonymous key can be used in all operations, such that users do not need to register in the *DeepBlue Server* before using it. The anonymous user has access to the public data available in *DeepBlue Server* and all its data selection and manipulation operations.

DeepBlue has a *super user* account that is created when a *DeepBlue Server* is initialized for the first time. This account is used for adding and removing users. The *super*

user account has access to all *DeepBlue Server* data and operations and it must be used carefully.

4.2.4 Entities

An entity is any content in the *DeepBlue Server* that can be individually accessed or named, for example, epigenomic experiments, genomic annotations, metadata content, biological sources, epigenetic marks, column types, and *Gene Ontology* (GO) terms.

These entities are categorized into two groups: data and metadata entities. The data entities are divided in several sub-groups: experiments, genomic annotations, gene models, gene expressions, and DNA sequences. These entities are used to store their respective (epi)genomic data.

The data entities are described using metadata that is a set of mandatory and optional terms organized in a key-value pairs data structure. For example, an experiment metadata consists of a unique name, the genome assembly name, the epigenetic mark, the sample ID, the technique, the project, the file format (the columns), and a set of optional key-value pairs. Annotation metadata consists of a unique name, genome assembly name, a description, the file format, and a set of optimal key-value pairs. The gene model metadata is also composed of a unique name, genome assembly, and a description. Similar metadata content applies to other data entities.

Metadata entities can also describe other metadata entities. For example, a sample is formed by a biosource name, and optional key-value pair terms that describe this sample content. Another example is the experiments and annotations content format, a list of columns describes this data content, each one representing a column of the original input file, and each column contains a name, data type, and other mandatory information.

For enforcing consistency in the metadata content, each mandatory metadata value is restricted to terms that were registered in their respective controlled vocabularies. For example, it is only possible to use the genome names that were previously recorded in the genomes controlled vocabulary. The same applies to other mandatory metadata fields: epigenetic marks, biosources, experimental techniques, projects, and columns. New terms can be easily added using their appropriate metadata insertion operation. Therefore, the *DeepBlue Server* can cope with new genomes, epigenetic marks, and biological sources.

4.2.5 Metadata Entities

The DeepBlue metadata entities are described in the following sections.

Genome

In the *DeepBlue Server*, a *Genome* refers to a specific genome assembly, for example *hg19* or *GRCh38*. It contains the genome assembly name, a description, and a list of chromosomes containing their names and lengths.

BioSource

In the *DeepBlue Server*, a *BioSource* is used to name the biological source (cell line, cell type, tissue, or organ) of a given *Sample*. The *BioSource* names used in *DeepBlue* follows the *IHEC* policy and are imported from the *Cell Type Ontology* (CL) (Bard *et al.* 2005), *Experimental Factor Ontology* (EFO) (Malone *et al.* 2010), and *UBERON* (C. J. Mungall *et al.* 2012).

The ontology terms are imported together with their synonyms and hierarchy. A *BioSource* term can be used to obtain all experiments related to its sub-terms. As an example, Figure 4.5 shows the hierarchy of the term *blood* in the *UBERON* ontology. In this ontology, the term *blood* is less specific and encompasses *umbilical cord blood*, *arterial blood*, *venous blood*, *capillary blood*, and it is more specific than *organism substance*. Using the hierarchical content, *DeepBlue Server* users can access experiments from different biological sources using the more general terms, for example, using the term *blood*, users can access all its more specific terms easily. As the *DeepBlue Server* also imports the synonyms of the terms, it is possible to access the term *blood* using one of its synonymous terms: *vertebrate blood* and *portion of blood*, while the *DeepBlue Server* manages the relationships between the names automatically.

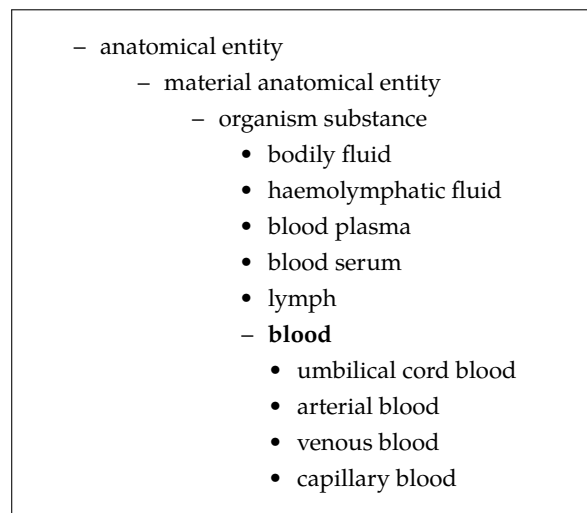


Figure 4.5: Example of the hierarchy of an *UBERON* term. In the *UBERON* ontology, the term *blood* is more general than *umbilical cord blood*, *arterial blood*, *venous blood*, *capillary blood*, and more specific than *organism substance*.

Sample

In the *DeepBlue Server*, a *Sample* refers to a biological sample of an epigenomic experiment. A sample mandatorily contains a biosource and it can additionally be annotated with optional key-value pairs. Even though all fields with the exception of *BioSource* are optional, the *DeepBlue Server* indexes them, allowing users to find specific samples, and consequently, epigenomic data. Figure 4.6 presents an example of a *Sample* imported into the *DeepBlue Server*.

BioSource	neutrophilic myelocyte
BIOMATERIAL_PROVIDER	Sanquin Amsterdam
BIOMATERIAL_TYPE	Primary Cell
CELL_TYPE	neutrophilic myelocyte
DISEASE	None
DONOR_AGE	80 - 85
DONOR_ETHNICITY	NA
DONOR_HEALTH_STATUS	Healthy
DONOR_ID	BM060814
DONOR_REGION_OF_RESIDENCE	-
DONOR_SEX	Female
SAMPLE_ID	ERS640348
SAMPLE_NAME	S00VCUH1
SPECIMEN_PROCESSING	-
SPECIMEN_STORAGE	-
TISSUE_TYPE	bone marrow

Figure 4.6: Example of a *DeepBlue Server* Sample.

Epigenetic Mark

In the *DeepBlue Server*, an *Epigenetic Mark* refers to the epigenetic mark that is the subject of the epigenomic experiment. The most common types of epigenetic marks are the DNA methylation, histone marks, transcription factors binding sites, and chromatin accessibility. The *DeepBlue Server* also uses RNA and its variants as epigenetic marks, as well as *Chromatin State Segmentation (CSS)*. In short, the *DeepBlue Server* groups all information about chromatin modifications, status, and product (e.g., RNA) as epigenetic marks. The goal is to simplify the usage of the data for the user by reducing the metadata complexity.

Technique

In the *DeepBlue Server*, a *Technique* refers to the experimental technique used in the epigenomic experiment. The metadata of an experiment contains its technique because, as shown in the Section 2.2, it is fundamental to define how the data was obtained and how it can be compared to other datasets.

Project

In the *DeepBlue Server*, a *Project* refers to the project that generated the epigenetic data file, for example, *DEEP* or *BLUEPRINT*. Projects can be public or private, where public projects can be accessible by all the users and private, only by its members.

Column Type

In the *DeepBlue Server*, a *Column Type* refers to the columns in the original experiment or annotation data file. A column type is composed of a name, description, and data type. Figure 4.7 lists the data types that can be used in the column types. It is important to distinguish *column types*, which are the columns used for describing an experiment or annotation file format, to their respective *data types*, which are the column content type.

For example, the *column type* *CHROMOSOME* specifies that the content of this column are of the *data type* *string*.

string	any text, usually for defining a name. Columns example: <i>CHROMOSOME</i> , <i>NAME</i>
integer	a value, usually for defining a count or position. Columns example: <i>START</i> , <i>END</i>
double	a real value, usually for defining scores and proportions. Column example: <i>VALUE</i> , <i>SCORE</i> , <i>SIGNAL_VALUE</i>
range	a range between two real values.
category	a list of available strings. Column example: <i>STRAND</i>
calculated	a short script that calculates the column value at run-time.

Figure 4.7: DeepBlue column data types. This table presents the data types that can be used in a *DeepBlue Server* *column type*.

For example, in the experiment format “*CHROMOSOME*, *START*, *END*, *NAME*, *SCORE*, *STRAND*”, the column type *CHROMOSOME* is a string, *START* and *END* are integers, *NAME* is a string, *SCORE* is a double, and *STRAND* is a category accepting the values “+”, “-”, and “.” (dot).

If, during the process of data import, the column content does not match the defined type, for example, a string where it is supposed to be an integer, the entire data file import process is canceled. This rigorous rule is fundamental for improving the quality of the data and metadata stored in the *DeepBlue Server*.

Gene Ontology

The *DeepBlue Server* uses GO (Ashburner *et al.* 2000; Consortium 2018) terms for annotating the imported genes. Users can select and filter genes by their GO terms, as well as perform enrichment analysis using this metadata content (Section 4.5.7). In contrast to *BioSources*, GO terms are imported without their hierarchical information.

4.2.6 Data Entities

As listed before, the *DeepBlue Server* has five types of data entities: experiments, annotations, gene models, gene expressions, and DNA sequence. Experiments contains the data (i) extracted from epigenomic experiments, e.g. DNA methylation or *ChIP-seq*; (ii) derived from existing epigenomic experiments data, e.g. *CSS*, or (iii) related to the epigenetic state of the cell, e.g. *RNA-seq* data. Annotations are the data used to augment the information of the genome, e.g. CpG island and repetitive regions.

In the *DeepBlue Server*, a gene model is a collection of the identified genes for a specific genome assembly. The *DeepBlue Server* imports the gene models from GENCODE (Harrow *et al.* 2012). The gene expression is a gene-centric expression data, usually in *fragments per kilobase of transcript per million mapped reads* (FPKM) or *transcript per million* (TPM) values. It uses a gene model for converting the gene-centric expression values to

genomic regions. The DNA sequence is the genome sequence that is accessed in operations for finding DNA patterns or can also be retrieved by meta-fields (Section 4.5.5).

The experiments and annotations are composed of a set of genomic regions, organized into chromosomes, and a genome. Each region contains its genomic location (*chromosome*, *start*, and *end*). It may also contain the region strand which contains the region direction.

The experiment metadata is formed by a unique name, the genome assembly name, the epigenetic mark, the sample ID, the technique, the project, the file format, and a set of optional key-value pairs. Annotation metadata consists of a unique name, genome assembly name, a description, the file format, and a set of optional key-value pairs.

The metadata field *format* describes the experiment or annotation regions' columns. For example, the format: "*CHROMOSOME, START, END, NAME, SCORE, STRAND*" specifies that all rows of this experiment or annotation have these six column types. As previously described, a column type must be registered in the column types controlled vocabulary before being used.

The metadata used to describe epigenomic datasets from different projects are heterogeneous: epigenomic experiments from different consortia have different organizations and contents. For handling this issue, the *DeepBlue Server* organizes experiments, annotations, and gene models metadata by mandatory and optional fields. The mandatory fields represent a subset of all available metadata, and they are the minimum required information for describing an epigenomic experiment and its data set. The provided metadata content that cannot be mapped to these fields is stored in the *extra-metadata* content, a key-value map that accepts any values, and allows the *DeepBlue Server* to save all available information about the entity.

All *DeepBlue Server* entities have an unique *ID*. The *IDs* are formed by a prefix with one or more letters, given by the entity type, and a number as suffix. For example, all experiments *IDs* start with an *e*, which is followed by a number. Table 4.1 shows some prefixes of the *DeepBlue* entities *IDs*. The operation *info* can be used for obtaining the information and metadata about an entity *ID*.

Experiment	<i>e</i>
Annotation	<i>a</i>
Genome	<i>g</i>
Gene model	<i>gs</i>
User	<i>u</i>
Biosource	<i>bs</i>
Sample	<i>s</i>
Technique	<i>t</i>
Project	<i>p</i>
Gene	<i>gn</i>
Column Type	<i>ct</i>
Operation	<i>q</i>
Processing Request	<i>r</i>

Table 4.1: Example of DeepBlue Server entities IDs prefixes.

4.3 Storing and Retrieving Epigenomic Data and Metadata

The *DeepBlue Server* contains a system for storing and handling (epi)genomic data. This complete data storage system can be divided into three parts: (i) storage system - how the data is stored; (ii) data model - how the data is organized; (iii) data access - how the data is retrieved. This section dissects the *DeepBlue Server* storage system and its components.

4.3.1 Storage System

The *DeepBlue Server* uses a MongoDB database⁹ for storing its entities: metadata and data content. The MongoDB database software was selected due to its horizontal scalability, that is the ability to increase the computational capacity by connecting multiple hardware or software entities so that they work as a single logical unit. MongoDB provides horizontal scalability through the use of sharding. MongoDB also provides a flexible data model centered on the concept of *documents* in which it is possible to include arbitrary information as key-value pairs.

In a classical relational database it would be challenging to store the data and metadata of the epigenomic experiments, since they are generally not homogeneous and structured differently for each of the contributing consortia. Using MongoDB allows the inclusion of standardized properties common to all data sources as well as additional information specific to each dataset. The *DeepBlue Server's* data and metadata are stored in MongoDB as documents, following the BSON specification. BSON¹⁰ is binary representation of the JSON format, which is a lightweight data-interchange format. These *documents* are grouped in *collections* that have a behavior similar to *SQL* database tables. Figure 4.8 shows a collection with three exemplary BSON documents.

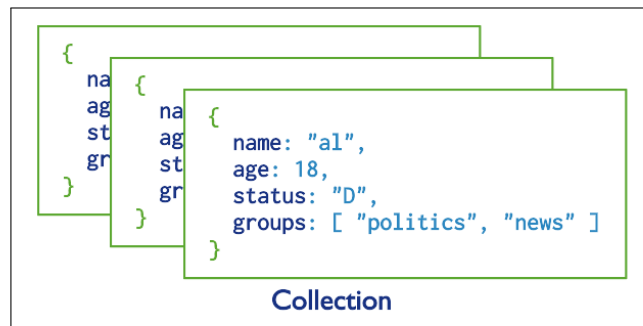


Figure 4.8: MongoDB collections with documents. Source: <https://docs.mongodb.com/manual/core/databases-and-collections/>.

MongoDB also provides other features that align with the *DeepBlue Server* needs, such as: (i) bulk insertions —it is possible to insert a set of data in a single and atomic operation, being essential for adding the epigenomic data efficiently; (ii) data distribution (sharding) —dividing the data into sections and storing these sections on different MongoDB servers, performing horizontal scaling; (iii) handling terabytes of data —it is

⁹ <https://www.mongodb.org>

¹⁰ <http://bsonspec.org/>

able to cope with vast amounts of epigenomic data, allowing the *DeepBlue Server* to store current and future epigenomic data.

When starting the *DeepBlue Epigenomic Data Server* project, the MongoDB database was compared with other *noSQL* databases such as *CouchDB*¹¹ and *Redis*¹². For each database, its scalability, flexibility, API usage, installation process, and the possible use with epigenomic data were analyzed. After the evaluation, MongoDB was chosen because of its horizontal scalability through sharding, flexible data model based on documents, an API with several language drivers, like *Python* and *C++*, and easy installation and use.

As presented in Section 4.2.1, the MongoDB database is accessed through the *Database Access* component. This component abstracts and facilitates the use of the *MongoDB C++ Driver*. In principle, it is possible to connect another database that implements the same document-oriented storage system in place of MongoDB, but currently, *DeepBlue Server*'s source code is strongly coupled with the MongoDB C++ driver code.

4.3.2 Data Model

Section 4.2.4 introduced the *DeepBlue* entities used to organize the (epi)genomic data and metadata information. Figure 4.9 gives an overview of the *DeepBlue Server Data Model*: experiments and annotation metadata are composed of their names, controlled vocabulary terms, and *extra-metadata*. The experiments and annotations data are composed of a set of regions, where each region is linked to the corresponding metadata by the dataset *ID* and each region contains a start and end, as well as the content of the data file columns. The regions are grouped and stored in their respective genome and chromosome collections (further details in Section 4.3.2). Genes from the gene models and the DNA sequence are stored separately from the other (epi)genomic data, in their own collections. All this information, data and metadata, must be serialized from and to BSON documents for being inserted into the database and thus available to the users, which is explained in the following sections.

4.3.3 Metadata serialization

The *DeepBlue Server* creates a BSON document for each metadata entity. These documents are grouped into collections, having one collection for each type of metadata. For example, the *DeepBlue Server* has the following collections for storing the metadata entities: *genomes*, *biosources*, *epigenetic_marks*, *projects*, *samples*, *techniques*, and *column_types*. Figure 4.10 shows an example of *BioSource* stored into the MongoDB database: as previously described in Section 4.2.4, each *BioSource* has an *ID*, name, description, *ID* of the user responsible for its inclusion, and *extra-metadata* containing the additional info that does not fit in the other fields.

Two extra fields were included in the document by the *DeepBlue Server*: *norm_name* and *norm_description*. The *norm* prefix means normalized, where the text was transformed to a normalized text representation. This transformation involves: removal of white spaces and special characters and transformation of the remaining text to lower case, for example, the text *DNA methylation* becomes *dnamethylation*. Normalized text is

¹¹ <http://couchdb.apache.org/>

¹² <https://redis.io/>

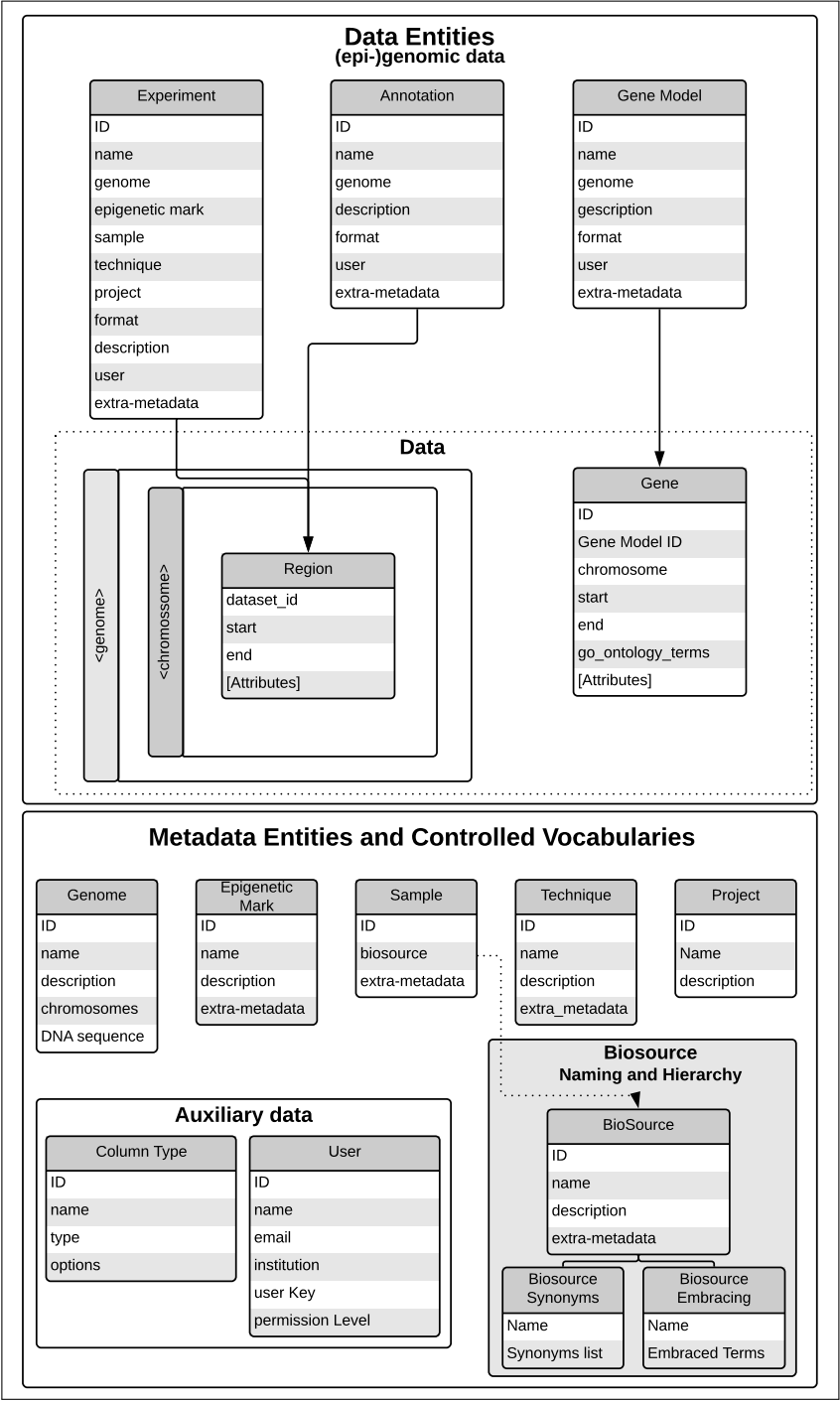


Figure 4.9: DeepBlue Server Data Model: The experiments and annotation metadata are composed of their names, controlled vocabulary terms, and *extra-metadata*. Experiments and annotations data are composed of region-sets, where its regions are grouped by chromosomes, and then by genomes. Each region is linked to the corresponding metadata by its *ID* field and contains a start and end, as well as additional attributes found in the data files. Links between experiment and annotation metadata fields to controlled vocabularies are omitted for clarity.

```

1 {
2   "_id": "bs1",
3   "name": "prostate duct",
4   "norm_name": "prostaduct",
5   "description": "The minute canals that pass the prostatic secretions to the thra.",
6   "norm_description": "theminutecanalsthatpasstheprostaticsecretionstotheurethra",
7   "extra_metadata": {
8     "comment": "",
9     "namespace": "uberon",
10    "ontology_id": "UBERON:0002485",
11    "url": "http://purl.obolibrary.org/obo/UBERON_0002485"
12  },
13   "user": "u2"
14 }

```

Figure 4.10: Example of BioSource document stored into the database.

used for performing searches in the database: when a user executes a search, its input text is also normalized and then compared to the normalized text stored in the metadata fields. In this way, the searching component can find the searched content even when the input differs in details like the letter case, empty spaces, or special text marks (e.g., commas and dots). Besides text normalization, the *DeepBlue Server* uses the MongoDB *index-text*¹³ capacities to index and full-text search all its metadata.

4.3.4 Storing epigenomic data

The (epi)genomic data insertion is composed of three steps: (i) parsing the file content to an internal data structure; (ii) converting the data structure to a BSON format; (iii) inserting the BSON data into the appropriate database collection. Each of these steps is performed by a different component. However, the user only needs to use the insertion operations provided by the *DeepBlue Server* API and is not concerned with these details.

The API operations *add_experiment* and *add_annotation* expect that the input data are in BED or WIG file formats. In the first step, the parser component reads the input data and converts it to an internal representation. Subsequently, this representation is converted to a list of regions. WIG files can be converted to an optimized format that requires less disk space.

The *DeepBlue Server* stores the regions as BSON documents in the database. In initial versions, the *DeepBlue Server* stored each region as an individual BSON document because it was the most straightforward way for storing such data. But due to the overhead generated by storing each region individually, especially in signal data files which are composed of million of regions, the regions storing system was optimized for storing multiple regions in a single document.

As the initial version is still used for storing genes from gene models, both versions are explained in the following sub-sections. For the sake of simplicity, the initial version is explained first, and then the optimized version. The Appendix A.2.1 contains the collections size using the initial and optimized version.

¹³ <https://docs.mongodb.com/manual/core/index-text/>

Serializing epigenomic regions

Initially, the *DeepBlue Server* serialized each region as a separate BSON document. This document contains the region *dataset ID*, *start*, *end*, and the region attributes from the file columns of the BED format, or one attribute, named *VALUE* for the signal data. Figure 4.11 depicts a region from a BED document, which is converted to the BSON document during the input data processing. The converted BSON document is shown in Figure 4.12. The region BSON documents are stored in collections defined by their entity (experiment or annotation), genome, and chromosome. In this way, it is not necessary to explicitly include the genome and chromosome in each BSON document that represents a region.

1	chr1	12372	12731	Region 1	0.721	+
---	------	-------	-------	----------	-------	---

Figure 4.11: Example of part of a BED file. The file columns are: *CHROMOSOME*, *START*, *END*, *NAME*, *SCORE*, and *STRAND*.

1	{
2	"_id": "region1",
3	"DATASET": 210693,
4	"START": 12372,
5	"END": 12731,
6	"NAME": "Region 1",
7	"SCORE": 0.721,
8	"STRAND": "+"
9	}

Figure 4.12: Example of BSON document containing the information of a Region. The genome and chromosome are implicit in the collection name where this region is stored, for example, collection *hg19.chr1*.

The BSON documents are stored with their *keys* and *values*, the keys became redundant¹⁴ and use a substantial amount of memory. For this reason, the *DeepBlue Server* compresses the *key* names, storing the minimal necessary name. For example, the *key* *DATASET* becomes *D*, the *START* becomes *S*, *SCORE* becomes *S1*, and *STRAND* becomes *S2*. An auxiliary collection stores the *keys* and their compressed names. Figure 4.13 shows a BSON document using compressed keys. Comparing the disk space required: the document in Figure 4.12 occupies 115 bytes and the compressed document in Figure 4.13 occupies 93 bytes, which corresponds to a decrease in size of approximately 20%.

Database systems provide data indices for optimizing data retrieval. MongoDB offers three types of indexes: full-text, hash, and B-trees. The *DeepBlue Server* uses full-text for indexing the metadata text, hash for categorical indexing data, and B-trees for indexing epigenomic data. These indices optimize the access to the stored epigenomic regions, but also generate overhead in disk and memory utilization.

DeepBlue uses database indices extensively for optimizing data retrieval. The regions are indexed by their unique identifier, by the start and end, and genomic location start

¹⁴ An improvement in the version 3.4 from MongoDB facilitated that the key compression is automatically performed.

```

1  {
2    "_id": "region1",
3    "D": 210693,
4    "S": 12372,
5    "E": 12731,
6    "N": "Region 1",
7    "S1": 0.721,
8    "S2": "+"
9  }

```

Figure 4.13: Example of BSON region with compressed key.

and end. MongoDB does not provide a direct way of calculating the memory and disk overhead generated by the index usage, but estimates show that each individually stored region uses 140 bytes on disk and memory to keep the three previously listed indexes. Thus, a single region document presented in Figure 4.14 requires 66 bytes for the region itself and 140 bytes for the indices, i.e. 200 bytes in total. Considering that a single signal file contains between 1,000,000 to 100,000,000 regions, the required disk space is between 200 megabytes to 18 gigabytes for a single file, where 13 gigabytes must stay in the main memory, which is impractical. For reducing these requirements, the *DeepBlue Server* uses different techniques that are explained in the following sub-section.

```

1  {
2    "_id": "region1",
3    "S": 52126,
4    "E": 57323,
5    "V": 1.27
6  }

```

Figure 4.14: Example of BSON document containing the information of a Region compressing the key names.

Data Blocks

Storing individual regions in the BSON format provides numerous benefits. The implementation is straightforward, as is data retrieval: the filtering of regions is fully executed by the database, retrieving only regions that match the given criteria. Furthermore, counting regions that match the input query can be performed directly in the database. This approach works well for peak files, but it is not practical for signal data due to the overhead for the indices.

To improve data indexing, the *DeepBlue Server* uses *data blocks* for handling signal data efficiently. Rather than storing each region individually, DeepBlue builds an array of continuous regions, named data block (Figure 4.15). Data blocks are compressed using the LZ0¹⁵ algorithm, which is a lossless data compression algorithm that is focused on decompression speed. Thus, compressing the data blocks saves valuable disk space without compromising data retrieval performance. A BSON document acts as an envelope of these regions, containing the block's information, such as start, end, count, and content type. This envelope is stored in the database regions collection respective to

¹⁵ <http://www.oberhumer.com/opensource/lzo/>

its genome and chromosome, and it is accessible by the *DeepBlue Server* data retrieval component.

```
{
  "S": 52100,
  "E": 52150,
  "C": 5
  "D":
    [
      {
        "S": 52100,
        "E": 52110,
        "V": 1.27
      },
      {
        "S": 52110,
        "E": 52120,
        "V": 1.20
      },
      {
        "S": 52120,
        "E": 52130,
        "V": 1.13
      },
      {
        "S": 52130,
        "E": 52140,
        "V": 1.15
      },
      {
        "S": 52140,
        "E": 52150,
        "V": 1.14
      }
    ]
}
```

Figure 4.15: Regions Data Block: The envelope document contains the identifier of the dataset, start and end positions of the block, and the count of regions. For clarity, internal attributes like the type of the stored regions, dataset *ID*, block size, and the flag for informing if block is compressed were omitted.

The *DeepBlue Server* also uses data blocks in peak regions, even though this results in a smaller gain when compared to the signal data. Besides the impressive reduction in index size, this strategy also improves data retrieval performance. The reason is that MongoDB has access to all continuous regions in the same location, and these regions are sent as an atomic data block to the *DeepBlue Server*, without the need to gather the information across the database. Furthermore, the transfer of the region-set data from MongoDB to the *DeepBlue Server* is optimized because the block content is compressed.

Gene models

The *DeepBlue Server* imports the gene models from GENCODE using the operation *add_gene_model*. This operation parses the gene model file content, which must be in GTF format and creates a BSON document for each gene, which is stored separately, rather than in blocks, in the genes collection. The reason for independently storing each gene is that genes are accessed individually or by small groups using their identifiers and names, not their genomic location.

Due to the various forms of accessing genes, the *DeepBlue Server* indexes the gene collections with more than ten indices. Each gene-model has approximately 60,000 genes/regions, totalling 266,711 genes for currently four imported gene models, using 255 megabytes for storing the documents and 450 megabytes for the indices, a very small fraction of the total data stored in the *DeepBlue Server*.

Quantified gene expression data

Together with the RNA data stored as signal, the *DeepBlue Server* also provides the expression data quantified by gene, not by genomic location. This data contains the *FPKM* and *TPM* of the expressed genes together with other information provided by the quantification tool, which can be either: Cufflinks (Trapnell *et al.* 2010), Grape (Knowles *et al.* 2013), or Salmon (Roberts and Pachter 2013). Due to the small amount of RNA data quantified by genes, where it is currently 40 millions quantified genes that use 20 gigabytes for the data and index, and the access being made individually through the gene identifiers, data blocks are not used and each quantified gene expression value is stored in a single BSON document, similarly to the individual genes from the gene models.

4.3.5 Genomic data storage

The *DeepBlue Server* provides genomic sequence data for helping with the analysis and integration of epigenomic data. These sequences are used for performing motif analysis or motif search. For example, it is possible to access all genomic locations where such a motif (given in the form of a regular expression) matches the DNA sequence, or to count how many times a motif occurs in the given genomic region.

The genomic sequences are inserted into the *DeepBlue Server* using the operation *upload_chromosome*, which requires a genome, chromosome, and a genomic sequence in the FASTA format, with the DNA sequence of the selected chromosome. Each genome, chromosome, and genomic sequence triplet is stored in the MongoDB database using its *GridFS*¹⁶ functionalities, which provide means for inserting each genomic sequence as an individual file and for retrieving portions of this file.

4.3.6 Epigenomic data insertion

The epigenomic data insertion process is divided into the following steps:

1. Metadata content verification
2. Epigenomic data transformation
 - a) data parsing
 - b) regions creation
 - c) data blocks generation
3. Metadata and data insertion
 - a) metadata insertion
 - b) data blocks insertion
4. Return of the data identifier

The verification step (1) confirms that the metadata contains all necessary attributes and that these attributes are presented in their respective controlled vocabularies.

The epigenomic data transformation step (2) parses the data file (2.a), (2.b) builds the internal representation of the genomic regions, and (2.c) generates the data blocks, compresses them, and includes them in the envelope BSON document. The parser performs lexical, syntactic, and semantic analysis of the data content. The lexical analysis checks if the text is valid, for example, confirming that the data contains only valid characters.

¹⁶ <https://docs.mongodb.com/manual/core/gridfs>

The syntax analysis verifies that the tab character separates its columns and checks if the data file has the number of columns defined in its format. Finally, the semantic analysis verifies that each row-column content obeys the selected data format. For example, neither an empty value nor a string are acceptable if the format specifies 'integer' or 'double'. The *DeepBlue Server* data parser is strict, and the entire process is canceled if a single content does not match the specified format.

In the next step (3), the metadata and data blocks are inserted into the database. During the insertion, the epigenomic data receives a unique identifier following the rules presented in Section 4.2.6.

For minimizing memory consumption of the data insertion process, the *DeepBlue Server* creates the data blocks while reading the data files and inserts them directly in the database. The data insertion process updates the metadata when the entire process is finished. If a problem occurs, e.g., invalid data is encountered, the already inserted data is deleted and the *DeepBlue Server* returns an error message. Otherwise, in the step (4) the (epi)genomic data identifier is returned.

4.3.7 Epigenomic data and metadata retrieval

The (epi)genomic metadata is retrieved when users wish to obtain information about a given experiment data file using its identifier, or when users want to select epigenomic data that matches some filtering criteria. Hence, for retrieving data, the respective metadata must be found and extracted.

The simplest case is when a user requests information about an experiment. In this case, the operation *info* is used with the experiment identifier. Internally, the *DeepBlue Server* executes a query in the MongoDB database for the experiment that contains the given identifier. The experiment BSON document content found is returned to the user.

In contrast to requesting epigenomic experiment metadata information, handling epigenomic data requires complex metadata queries and data retrieval procedures. This procedure is composed of three steps: (i) finding the relevant experiments in the experiments metadata collection; (ii) obtaining the data blocks that contain the required regions; (iii) extracting and filtering the regions from the data blocks, and returning them to the user. The following paragraphs detail this procedure.

The first step is to prepare a metadata filtering query, containing all user-defined filtering criteria, e.g. filtering by epigenetic mark, genome, and other metadata attributes, with internal criteria, e.g. which data the user has permission to access. This query accesses the experiments metadata collection, and the *DeepBlue Server* collects the identifiers from all experiments that match the query criteria for using in the second step.

In the second step, *DeepBlue* prepares a filtering query, specifying the location of the regions (start, and end) and the experiments IDs found in the first step. The data filtering query accesses each chromosome collection specified by the query, obtaining a set of BSON data blocks.

In the third step, the regions are extracted from the data blocks and filtered by the *DeepBlue Server* using the given genomic location ranges or by the regions' attributes, e.g., if a given column matches the filtering criteria ($P_VALUE \geq 0.06$, or $NAME == 5_Strong_Enhancer$). The resulting regions are converted to the internal *DeepBlue* region data structure, and then used in the subsequent operations.

4.4 DeepBlue Populator

The DeepBlue populator sets up the *DeepBlue Server* configurations, imports the annotations and experiments data and metadata, and ensures that the *DeepBlue Server* data is kept up to date. It executes all necessary operations independently and automatically, without the need of any manual intervention on the *DeepBlue Server*. It periodically examines the data sources of relevant epigenome projects and imports the data automatically. The populator is implemented in *Python* and has approximately 8,500 lines of code. Appendix A.2.2 provides additional implementation details and instructions for configuring and executing the populator.

The core responsibilities of the populator are: (i) setting up a new *DeepBlue Server* configuration, (ii) importing metadata entity terms, (iii) importing the (epi)genomic data, (iv) and keeping the data up-to-date. In this section, the following responsibilities are explained in more detail:

- Initiating the *DeepBlue Server* and create initial users
- Importing controlled vocabularies for epigenetic marks and techniques
- Importing BioSources from the ontologies
- Importing genomes
- Inserting genomic DNA sequences
- Inserting genomic annotations
- Inserting genes and gene ontologies
- Importing epigenomic experiments

Initiating the *DeepBlue Server* and create initial users

The primary responsibility of the populator is to initialize the *DeepBlue Server*. It executes the administrative operation *init* which creates the MongoDB database instance with its initial collections and required indexes. This operation also creates the administrator and anonymous accounts. The populator uses the administrator account for creating the *populator* user which is used in all populator operations, such as metadata and data insertion.

Importing Epigenetic Marks and Techniques controlled vocabularies

The populator source code contains a list of epigenetic marks that must be inserted into the *DeepBlue Server*. The histones from this list are imported from the *H1stome database* (Khare *et al.* 2011), the *Transcription Factor Binding Site (TFBS)* from the *ENCODE Project* and *ChIP-Atlas*. Other epigenetic marks, such as the DNA methylation, are imported from literature.

Importing BioSources from the ontologies

The populator imports the BioSources terms and their hierarchy from *Cell Type Ontology (CL)* (Bard *et al.* 2005), *Experimental Factor Ontology (EFO)* (Malone *et al.* 2010), and *Uber Anatomy Ontology (UBERON)* (C. J. Mungall *et al.* 2012) ontologies. These ontologies are

stored in the *Web Ontology Language* (OWL) format¹⁷, which needs to be processed and organized before being inserted into the DeepBlue Server.

During processing, the populator extracts term names, synonyms, and their hierarchy. While extracting term names and synonyms are relatively straightforward tasks, extracting hierarchy can be more complex due to the existence of different types of relationships. For example, the populator requires that the relationship between terms is expressed in the form of a term *A* being more specific than a term *B*. But ontologies provide the relationships in this form and also in the opposite form, where a term *A* is less specific than term *B*. Hence, it is necessary to verify how this information is provided and, if necessary, to correct the relationship information. For this, the populator keeps a blacklist of invalid and a whitelist of valid term relationships. The BioSources importing method is one of the most complex parts of the *DeepBlue Server* and its implementation is found in the *DeepBlue-Populator/src/owl_loader.py* source code file. Due to its independence from the main populator code, the OWL extractor and parser can be reused in other projects as well.

Importing Genomes

The populator imports mouse genome assembly *GRCm38* and the human genome assemblies *GRCh38*, *hg19*, and *hs37d5*. All genome assemblies are imported with their respective chromosome names and sizes. The populator uses the operation *add_genome* to include a new genome assembly. Internally, the *DeepBlue Server* creates a MongoDB collection where the regions data is stored for each genome and chromosome. Adding a new genome assembly to the *DeepBlue Server* is trivial, as only the name of the assembly and its chromosome names and sizes are required. Table A.1 in Section A.1 contains the source URLs of the imported genomes and their respective DNA sequences.

Inserting Genomes DNA sequences

The populator includes genomes' DNA sequences that can be accessed and analyzed directly in the *DeepBlue Server*. Each genome-chromosome pair is included individually using the operation *upload_chromosome*. This operation requires the genome assembly name, chromosome, and the DNA sequence in FASTA format. Internally, the *DeepBlue Server* verifies the DNA sequence content and its length.

Inserting (Epi)genomic Annotations

The populator includes commonly used annotations such as *GpG Islands*, *Repetitive Regions*, and the ENSEMBL regulatory build (Zerbino *et al.* 2015). It uses the operation *add_annotation* to include the annotation data. Additional annotations can be easily added in the *DeepBlue Server* using the same operation.

¹⁷ https://en.wikipedia.org/wiki/Web_Ontology_Language

Inserting Genes and Genes Ontologies

The populator imports genes from the *GENCODE* versions 19, 22, and 23 for human genome, and the *M1* and *M13* for the mouse genomes. Each imported *GENCODE* version, with all its genes, is organized in a *DeepBlue* entity collection named *gene-model*. Table A.2 in Section A.1 contains the source *URLs* of the imported genome assemblies.

The human genes are annotated with *GO* (Ashburner *et al.* 2000; Consortium 2018) terms that are used in regions enrichment by *GO* terms or for finding specific genes.

As *GO* data is mapped to protein identifiers rather than to gene identifiers, its import process requires pre-processing. The *GO* data import process is performed by the following steps: (i) obtaining the *GO* terms and their hierarchy from the *GO* OWL file; (ii) loading the *ID* mapping from the proteins *IDs* and *GO IDs*; and (iii) annotating the genes with the mapped *GO* terms. The implementation is available in the *DeepBlue-Populator/src/gene_ontology.py* source code file.

Importing the epigenomic experiments

Importing epigenomic experiments from the epigenomic projects¹⁸ is the main and most complex task of the populator. These steps compose the experiment insertion process:

1. Read the metadata from the different data sources, e.g., epigenomic project portals.
2. Store each dataset's metadata as a BSON document in the MongoDB populator database.
3. Query these documents by their experiment data (signal or peak data) and by their metadata.
4. Convert documents to a general DeepBlue metadata content using a project-specific mapper¹⁹. The mapper extracts the project-specific metadata information from the stored document and builds a DeepBlue's metadata representation.
5. Download and convert the experiment data into a textual format, e.g. from bigBed to BED or WIG to WIG or *bedgraph* file formats.
6. Insert the experiment data and metadata into the *DeepBlue Server* using the operation *insert_experiment*.
7. Repeat this process for all configured data sources and for all experiments of each data source that matches the insertion criteria.

Extracting the metadata content from the project metadata files is a cumbersome task because the metadata file format varies from project to project. For example, the initial *ENCODE* (ENCODE Project Consortium 2004; ENCODE Project Consortium 2012) metadata data files²⁰ were key-value pairs with a simple description of the experiment data file. Nowadays, *ENCODE* provides a powerful API (Sloan *et al.* 2015) for querying its data, which is dealt with by a special class in the populator²¹. The BLUEPRINT

¹⁸ List of projects in Appendix A.1

¹⁹ Implementation of the project-specific mapper are in *DeepBlue-Populator/src/datasources*

²⁰ Implementation in *DeepBlue-Populator/src/encode_repository_ftp.py*

²¹ Implementation in *DeepBlue-Populator/src/encode_repository.py*

Epigenome project (Adams *et al.* 2012; Martens and Stunnenberg 2013) provides a table file with the metadata of its experiment files, thereby requiring a different implementation²² for accessing and handling this metadata. Similarly *DEEP*, RoadMap and ChIP-Atlas data also needs special classes for handling metadata. Fortunately, the *IHEC* Data Portal (Bujold *et al.* 2016) has a standard metadata format, where, a priori, all *IHEC* member projects' have the same metadata standard. As noted before, after reading the metadata from the data source, the populator stores a BSON document containing the metadata and data file localization for each found dataset file.

It is straightforward to extend the populator to import data from a new epigenomic project. It is only necessary to provide a method for accessing the experiments metadata, transforming this metadata to the *DeepBlue Server* representation, and downloading the experiment data.

4.5 Application Programming Interface (API)

The API is a central piece of the *DeepBlue Epigenomic Data Server*. The *DeepBlue* API comprises a comprehensive set of operations provided by the *DeepBlue Server*. The primary goal of the API is to offer a *programmatic interface* to the epigenomic data and metadata stored in the *DeepBlue Server*, as well as to allow users to access and handle the data efficiently. The API can be used anonymously with access to all public data. Besides, users can create an account to have access to their operations history, to upload their own data, and to access or share privately managed data in their *DeepBlue Server* workspace.

The API is accessible by all programming languages that support the XML-RPC or *RESTful* protocols, which includes *Python*, *R*, *Matlab*, *Java*, *Perl*, *C++*, *PHP*, and *JavaScript*. The next chapters and sections present examples that demonstrate the *DeepBlue Server* API versatility: the *DeepBlue Web Portal* (Chapter 6) was developed using *HTML*, *JavaScript*, and *PHP*; the visual exploratory tool *DIVE* (Chapter 7) was developed in *HTML* and *TypeScript*; the Batch Effect Analysis Tool (Section 5.3.4) was developed in *R*; and many use cases were developed using *Python* and *R*. Furthermore, external groups also used the *DeepBlue Server* API for developing (epi)genomic data analysis tools, such as the BLUEPRINT Data Analysis Portal (J. M. Fernández *et al.* 2016), developed in *Perl*, and *deepTools* (Ramírez *et al.* 2016), developed in *Python*.

The *DeepBlue Server* API comprises two types of usage: administration and end user. The administration operations include operations for including new metadata terms, new experiments, removing data or metadata, creating projects, and registering users. These operations are executed solely by the Populator tool (Section 4.4) or by the *DeepBlue Server* administrator rather than by the user.

From the user point of view, the API operations can be divided into seven main categories: metadata information retrieval, list and search, selection, manipulation, enrichment, requesting, and downloading results. Table 4.2 summarizes the main operations found in these categories. Appendix A.4 contains the full list of API operations.

Access control is regulated with *user_key*. A *user_key* is a user's individual identification, and it is used in every request made to the *DeepBlue Server*. Users must keep their *user_key* private. The *DeepBlue Server* also provides a generic *user_key*, the *anonymous_key*, that can be adopted by any researcher who does not wish to create a personal

²² Implementation in *DeepBlue-Populator/src/blueprint_repository.py*

Category	Operation	Description
Entity Information	<i>info</i>	Obtain the metadata information about an entity
List & Search	<i>list_genomes</i>	List all registered genomes
	<i>list_biosources</i>	List all registered biosources
	<i>list_samples</i>	List all registered samples
	<i>list_epigenetic_marks</i>	List all registered epigenetic marks
	<i>list_experiments</i>	List all available experiments
	<i>list_annotations</i>	List all available annotations
	<i>faceting_experiments</i>	Summarize experiments by metadata
	<i>is_biosource</i>	Verify if the given name is a valid BioSource
Selection	<i>search</i>	Perform a full text search
	<i>select_regions</i>	Select regions from experiments
	<i>select_experiments</i>	Select regions from experiments
	<i>select_annotations</i>	Select regions from annotations
	<i>select_genes</i>	Select genes as regions
	<i>select_expressions</i>	Select genes expression as regions
	<i>tiling_regions</i>	Generate tiling regions
Manipulation	<i>input_regions</i>	Upload and use a small region sets
	<i>aggregate</i>	Aggregate and summarize regions
	<i>filter_regions</i>	Filter regions using their attributes
	<i>flank</i>	Generate flanking regions
	<i>intersection</i>	Filter overlapping regions
	<i>merge_queries</i>	Merge two region-set
Enrichment	<i>enrich_regions_overlap</i>	Enrich regions by overlaps count
	<i>enrich_regions_fast</i>	Enrich regions using bitmap representations
	<i>enrich_regions_go_terms</i>	Enrich regions with gene ontology terms
Request	<i>count_regions</i>	Count selected regions
	<i>score_matrix</i>	Request a score matrix
	<i>get_regions</i>	Request the selected regions
Download	<i>get_request_data</i>	Obtain the requested data

Table 4.2: The *DeepBlue Server* API categories and main operations for each category. A typical workflow starts with experiment search, followed by data selection. Optionally, selected data can be manipulated before being counted or retrieved. The results can be downloaded as a formatted table, or a score matrix.

account. With the *anonymous_key*, users have access to all public datasets and can perform all non-administrative operations. Additionally, registered users can: insert and access private data, share private data with other users through private projects, access previously executed operations and requests. Registered users also have permission for longer workflows execution time and higher memory usage.

All API operations return a value pair: (*status*, *result*). For successful operations, the variable *status* contains the string "okay" and *result* contains the operation result. If some error happened, *status* contains the string "error" and *result* contains an error description.

The API categories and the most frequently used operations are presented in the following sections. Not all operations are discussed, but a comprehensive list of the API operations is presented in Appendix A.4

4.5.1 Entities information operation

All *DeepBlue Server* entities, e.g., experiments, annotations, controlled vocabulary terms, and processing request, have a unique identifier, which can be used to obtain detailed information through the *info* operation. This operation is very versatile: it is used to obtain the metadata of a set of experiments, sample information, or the status of a processing request. In short, the *info* operation is the main source of metadata and status information in the *DeepBlue Server* for the users. It can be used with a single or a set of *IDs*, being optimized to return the information of thousands of different *IDs* in a feasible time frame of a few seconds.

4.5.2 Listing and searching operations

The *DeepBlue Server* stores a large number of controlled vocabulary terms. These terms are divided into entities used for annotating thousands of (epi)genomic datasets. Due to the large amount, the API provides listing and searching operations for finding metadata terms and/or the desired (epi)genomic data. The listing and searching operations are divided into three types: (i) listing, (ii) text-search, and (iii) faceting.

The listing operations query the *DeepBlue Server* for the entities that match the specified criteria. For example, the *list_genomes* operation returns a list of all genome assemblies registered in the *DeepBlue Server*. The same concept extends to other list operations, such as *list_experiments*, which lists all experiment files that match the given filtering parameters. Virtually, all entities have a *listing* operation: *list_genomes*, *list_epigenetic_marks*, *list_samples*, *list_techniques*, and *list_projects*.

The API also provides a set of operations that uses the *Levenstein distance* for finding entity names using similar names. For example, executing the operation *list_similar_epigenetic_marks* with the input *DNA meth* returns *DNA methylation*, or using *H3Kac* returns *H3K27ac* and other epigenetic marks with similar names. These operations are useful in visual interfaces for verifying the user input and suggesting correct names.

While *list_experiments* is used to obtain a list of all experiments that match the given filtering criteria, it does not group the experiments by their metadata content. For such purposes, the *faceting_experiments* operation allows users to list the desired experiments, and return a list of the selected experiments grouped by their metadata content. This

operation gives users an overview of the data available in the *DeepBlue Server* that match the searching criteria.

Listing 4.1 shows an output example of the *faceting_experiments* operation. This operation returns a dictionary containing metadata fields that annotate the experiments matching the filtering parameters. Each metadata field contains a list of terms and the counts of experiments that are annotated by this term.

```

1 >>> server.faceting_experiments("GRCh38", "peaks", None,
2   "CD14-positive, CD16-negative classical monocyte",
3   None, "chip-seq", "BLUEPRINT Epigenome", user_key)
4
5 # Result:
6 ['okay', {
7   'biosources': [['bs16177', 'CD14-positive, CD16-negative classical monocyte', 66]],
8   'epigenetic_marks': [['em100', 'H3K9/14ac', 1],
9     ['em64', 'H3K27me3', 10],
10    ['em67', 'H3K36me3', 7],
11    ['em79', 'H3K9me3', 11],
12    ['em70', 'H3K4me1', 13],
13    ['em72', 'H3K4me3', 10],
14    ['em60', 'H3K27ac', 14]],
15   'genomes': [['g7', 'GRCh38', 66]],
16   'projects': [['p2', 'BLUEPRINT Epigenome', 66]],
17   'samples': [['s10855', '', 12],
18     ['s10907', '', 7],
19     ['s10867', '', 5],
20     ['s10483', '', 7],
21     ['s10531', '', 8],
22     ['s10484', '', 5],
23     ['s10498', '', 8],
24     ['s10885', '', 7],
25     ['s10861', '', 5],
26     ['s10904', '', 2]],
27   'techniques': [['t5', 'ChIP-seq', 66]],
28   'types': [['', 'peaks', 66]]}
29 ]

```

Listing 4.1: Example of the *faceting_experiments* operation: it presents the result of the operation *faceting_experiments* filtering by the genome *GRCh38*, *peaks* experiments, biosource *CD14-positive, CD16-negative classical monocyte*, *ChIP-seq* technique from the BLUEPRINT Epigenome project. This operation returns a dictionary with the metadata entities used to annotate the experiments. Each dictionary key represents a metadata entity and the dictionary values contain a list of terms and count of how many experiments are annotated by this term.

The API also provides a *full-text search* operation that allows users to search in all *DeepBlue Server* data and metadata entities using a simple full-text entry, similarly to a Google search. This operation is built on top of the MongoDB's *Index-Text*²³. The input text may use special characters for qualifying the text. For example, a hyphen (-) in front of a word denotes that this word must not be contained in the indexed metadata, in opposite, single quotes (') must embrace the designed word for marking it as mandatory. The *search* operation returns a list of lists of results, where each list contains three elements: entity *ID*, entity name, and entity collection. Further information about the entity can be obtained using the *info* operation.

²³ MongoDB' Index-Text documentation: <https://docs.mongodb.com/manual/core/index-text/>

```

1 >>> server.search("'H3K27ac' 'blood' -CD4 'peak' 'Roadmap'", "experiments", user_key)
2 ['okay', [
3   ['e19426', 'E047-H3K27ac.narrowPeak.bed', 'experiments'],
4   ['e19788', 'E048-H3K27ac.narrowPeak.bed', 'experiments'],
5   ['e19421', 'E047-H3K27ac.gappedPeak.bed', 'experiments'],
6   ['e19439', 'E047-H3K27ac.broadPeak.bed', 'experiments'],
7   ['e19783', 'E048-H3K27ac.gappedPeak.bed', 'experiments'],
8   ['e19798', 'E048-H3K27ac.broadPeak.bed', 'experiments']]
9 ]

```

Listing 4.2: Example of the *search* operation: top five results of the full-text search for *H3K27ac*, *blood*, *-CD4*, *peak*, *Roadmap* in the collection *experiments*. The *search* operation returns up to 50 entries that are most similar to the query with their *ID*, name, and collection.

Besides the searching and listing operations, it is possible to verify if a given name is a valid BioSource term through the operation *is_biosource*. This operation receives a name and return "okay", if the term is a valid BioSource, or "error", if it is an invalid term.

The *preview_experiment* operation receives the name of an experiment, and directly returns the first 5 lines of this experiment's content. In this way, users can have a preview of the experimental data, without the need to retrieve the whole experiment through the *DeepBlue Server's* data processing workflow.

4.5.3 Server-side data processing workflow

The *DeepBlue Server* enables users to perform operations on (epi)genomic data efficiently, where its API is designed for operating and manipulating (epi)genomic region-sets directly on the server. The functionalities of the API include the following (epi)genomic data operations: selecting regions by their location or their dataset meta-data, filtering regions by their content, detecting overlaps, aggregating regions and providing summary statistics, searching for DNA motifs, and counting, retrieving, and enriching regions. When extracting the information of a region, the result can be downloaded in a tabular (BED file format like), or matrix format.

The *DeepBlue Server* operates on defined genomic sets of regions, which are selected by the selection operations. Each data selection execution returns a unique *ID*, which is used as input for the region-set data manipulation operations, for example: aggregating, filtering regions by their content, generating flanking regions, finding overlapping regions, or merging different region-sets based on additional criteria. Each of these operations returns a new unique *ID*, which are also used as input for following data operations. The concatenation of operations through their *IDs* allows users to build and execute complex (epi)genomic data processing workflows, where in the end, a request *ID* is used for downloading the final results.

In the end of the workflow building process, when a user sends a processing request, the *DeepBlue Server* processes it asynchronously. This means that rather than blocking the connection while waiting for the processing be finished, the server returns a request identifier (*request_id*), which is used to retrieve the request processing status and its result. The *request_id* is used in the *info* operation for obtaining the status of a processing request, and for obtaining the processed data through the operation *get_request_data*.

Figure 4.16 shows an overview of how the processing workflow is constructed and executed in the *DeepBlue Server*, where a typical workflow is composed of data selection (4.5.4), data manipulation (4.5.5), and data retrieval (4.5.6).

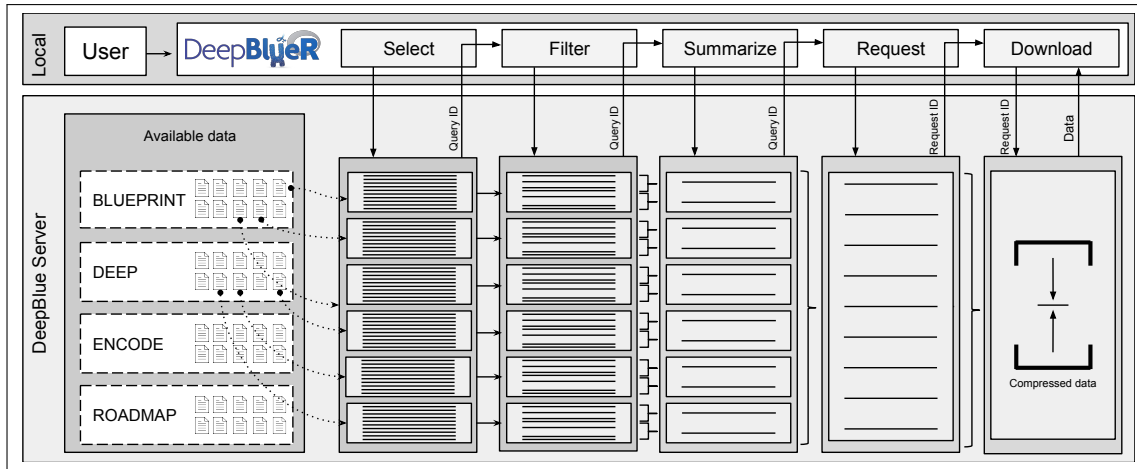


Figure 4.16: DeepBlue processing workflow: the *DeepBlue Server* facilitates combining data operations into a data processing workflow. For each operation, an *ID* is returned and the final data is accessible through the request *ID*. (Figure modified from Albrecht *et al.* 2017)

4.5.4 Data selection

The *DeepBlue Server* API provides operations for selecting the desired (epi)genomic data. In the following, these operations and other data selection methods are presented and explained.

Experiments data selection

As previously described, the *DeepBlue Server* workflow operates on a set of regions and the selection of such regions is the first step of any workflow. The operation *select_experiments* is the most straightforward experiment data operation, where users select the experiments by their names.

The operation *select_regions* is a more powerful variant where users can select all genomic regions associated to the metadata filtering criteria. This operation is very powerful, because it allows the selection of a large amount of (epi)genomic data in a simple operation. As an example, the source code `(status, q_h3k27ac_blueprint)= server.select_regions("", "GRCh38", "H3K27ac", samples_id, "ChIP-seq", "BLUEPRINT project", ["chr1", "chr2", "chr3"], None, None, "anonymous_key")` selects all regions annotated by the genome assembly *GRCh38*, epigenetic mark *H3K27ac*, and the sample *ID* stored in the variable *sample_id*, from from *BLUEPRINT* project, located in the chromosomes *chr1*, *chr2*, and *chr3*.

The command *select_regions* does not make any distinction between peaks and *signal* regions. For distinguishing the regions' type, the operation *query_experiment_type* is used in combination with the data selection operations: `(status, q_h3k27ac_blueprint_peaks)= server.query_experiment_type(q_h3k27ac_blueprint, "peaks", "anonymous_user")`, where the

query_id q_h3k27ac_blueprint_peaks references the subset of the *q_h3k27ac_blueprint* regions that are peaks.

Annotations data selection

The API provides the operation *select_annotations* for selecting annotation regions. This operation has similar behavior as the *select_experiments* operation, but selecting region-sets from *annotation* files rather than experiment files. For example, the source code: `(status, cpgs_id)= server.select_annotations("CpG Islands", "hg19", ["chr21", "chr22"], None, None, user_key)` selects the genomic regions of the *CpG Islands* annotation, from the genome *hg19*, located on chromosomes *chr21* and *chr21*.

Gene data selection

The *DeepBlue Server* also handles genes directly. Genes are selected with the operation *select_genes*. With this operation, users can select genes by their names, GO terms, and by genomic location. In this way, users can perform analysis on genes lists, GO terms, or genes located in a specific genomic location.

Processed gene expression data selection

The *DeepBlue Server* provides two types of gene expression data: (i) RNA experimental files, located within other (epi)genomic experiments, and (ii) RNA expression quantified by gene. Mapping projects usually provide this data in *FPKM* or *TPM* data format files, which do not contain genomic locations, but the *IDs* of the quantified genes. For this reason, these data files are stored and accessed differently by the *DeepBlue Server* API.

RNA expression data quantified by genes is accessed through the *select_expressions* operation. This operation has the following parameters: *expression_type*, a list of *sample_ids*, *replicas*, *identifiers*, and *gene_model*. The *expression_type* contains the type of expression data, but currently, only the type "genes" is available. The *sample_ids* are used to select the data via sample, facilitating the selection of the data of the same sample from other (epi)genomic experiments. The *replica* parameter inform which experiment replica must be used. The parameter *identifiers* allows the user to select the expression of individual genes by their *Ensembl IDs* or *ENSB* names. Finally, the *gene_model* parameter reports which gene-model is used to map the gene expressions to their genomic locations, a task that is completely transparent to the user. With exception of *gene_model*, the other parameters are optional and can be freely combined.

DNA motif regions

The *DeepBlue Server* API provides different methods for accessing and linking the (epi)genomic data to DNA sequences. For example, it is possible to access DNA sequences using meta-fields (Section 4.5.5), e.g. *@SEQUENCE* and *@COUNT.MOTIF*, or by the API operations such as *filter_by_motif*, which filter the regions that overlap to a DNA sequence motif. Furthermore, the operation *find_motif* constructs genomic regions in run-time that match a specific DNA sequence motif.

Tiling regions selection

Tiling regions are consecutive regions with the same size that span the complete genome. The *DeepBlue Server* uses tiling regions as bins to discretize (epi)genomic data, usually as result of aggregation operations. It is possible to generate tiling regions at run-time through the operation *tiling_regions*.

Uploading regions

The *input_regions* operation allows users to upload genomic regions for being used in the processing workflow. This operation has only two main parameters: the genome to which these regions belong, and the regions themselves. The input format is automatically deduced from the following possible list of formats:

- CHROMOSOME, START, END, SCORE
- CHROMOSOME, START, END, NAME
- CHROMOSOME, START, END, SCORE, NAME
- CHROMOSOME, START, END, NAME, SCORE, STRAND, SIGNAL_VALUE, P_VALUE, Q_VALUE, PEAK
- CHROMOSOME, START, END, NAME, SCORE, STRAND, THICK_START, THICK_END, ITEM_RGB, BLOCK_COUNT, BLOCK_SIZES, BLOCK_STARTS

After the regions inclusion, they are accessed through a *query ID*.

4.5.5 Data manipulation

As a counterpart to the previous (epi)genomic data selection methods, the *DeepBlue Server* provides a set of operations for manipulating data directly on the server. These operations are classified in: filtering, transformation, and statistics. Before diving deep in the data manipulation operation, this section first presents the meta-fields.

Meta-fields

The meta-fields are complementary operations used with the API operations. They support the tasks of manipulating and retrieving (epi)genomic data. Meta-fields are pseudo data columns that, rather than accessing an (epi)genomic data column, execute an operation in the context of the individual genomic region.

Table 4.5.5 presents all meta-fields available in the *DeepBlue Server*. This table separates the meta-fields by their category, such as, accessing the regions metadata, obtaining the correspondent region DNA sequence, obtaining the gene GO annotations, and calculating values during run-time. This table also shows what each meta-field returns.

The *Region information* meta-fields are used to obtain region-specific information: its length (@LENGTH) and the region strand (@STRAND). The *DNA sequence* meta-fields helps to obtain the DNA sequence (@SEQUENCE) associated with the genomic region and to count how many times a regular expression matches the corresponding region's DNA sequence (@COUNT.MOTIF). For example, the meta-field @COUNT.MOTIF((TATA|GAGA)) returns how many times the patterns TATA or GAGA appear in the region's DNA sequence. The *Genes and GO* meta-fields allow users to access genes and GO meta-data respectively. In similar way, *Experiment metadata* meta-fields provide means for accessing the regions' experiment metadata. For example, it is possible to obtain the experiment's name (@NAME) or the experiments biosource name associated to such region (@BIOSOURCE) directly. If the region does not belong to an experiment, only the @NAME meta-field returns a value, all other *Experiment metadata* meta-fields returns an

Category	Meta-field	Type	Return
Region information	@LENGTH	integer	Region's length
	@STRAND	string	Region's strand
DNA sequence	@SEQUENCE	string	Region's DNA sequence
	@COUNT.MOTIF	integer	Motif count in the Region's DNA sequence
Genes and GO	@GENE_ATTRIBUTE	string	Gene attributes
	@GENE_ID	string	Gene ID
	@GENE_NAME	string	Gene name
	@GENE_EXPRESSION	double	Gene expression value
	@GO_IDS	string	Gene ontology IDs
	@GO_LABELS	string	Gene ontology labels
Experiment metadata	@NAME	string	Entity (Experiment, Annotation) name
	@EPIGENETIC_MARK	string	Experiment's Epigenetic Mark
	@PROJECT	string	Experiment's Project
	@BIOSOURCE	string	Experiment's BioSource
	@GENOME	string	Experiment's Genome
	@SAMPLE_ID	string	Experiment's Sample ID
Aggregation Results	@AGG.MIN	double	Minimum aggregated value
	@AGG.MAX	double	Maximum aggregated value
	@AGG.SUM	double	Sum of aggregated values
	@AGG.MEDIAN	double	Median of aggregated values
	@AGG.MEAN	double	Mean of aggregated values
	@AGG.VAR	double	Variance of aggregated values
	@AGG.SD	double	Standard deviation of aggregated values
	@AGG.COUNT	double	Count of aggregated regions
Calculated values	@CALCULATED	string	

Table 4.3: Description of the *DeepBlue Server* meta-fields.

empty *string*. The *Aggregation Results* meta-fields are used for obtaining the results of the *aggregate* operation, which is presented further in this section.

Calculated values

With *Calculate columns* users can generate custom columns content at run-time. Calculated columns can be accessed by the meta-field @CALCULATED() or by creating a calculated column using the operation *create_column_type_calculated*.

The @CALCULATED() meta-field allows users to execute a short *script* that performs calculations on the current region at run-time. For example, the code @CALCULATED(
`return math.log(value_of('SCORE'))`) returns the log values of the column *SCORE*, and the code: @CALCULATED(
`em = value_of('@EPIGENETIC_MARK') if em == 'DNA Methylation' then return 'it is DNA Methylation' else return 'it is not methylation' end`) obtains the region epigenetic marks and returns a different text in case of it being *DNA methylation* or not.

The *scripts* are written in Lua programming language, because the *DeepBlue Server* has an embedded Lua (Ierusalimschy 2006) interpreter. For security reasons, the Lua code is executed in a sandbox environment, which means that users cannot access all Lua API and functionalities due to security issues, but still can execute the *mathematical*, *string manipulation*, and *types conversion* functions. A list of available commands is presented in Table A.10 in the Annex A.2.1 together with additional information about the Lua

interpreter. In addition, the sandbox protects the *DeepBlue Server* against abusive usage by limiting the interpreter memory usage and execution time.

Filtering (epi)genomic regions

The *DeepBlue Server* API provides operations for filtering the (epi)genomic data regions based on the region columns content and based on overlapping regions. In the following, both methods and associated operations are presented and examples are given.

Filtering by regions content

The *DeepBlue Server* API enables filtering regions by their content through the operation *filter_regions*. For example, this operation can be applied to DNA methylation data regions and filters the regions, which have a DNA methylation level higher or lower than a defined threshold. A similar approach can be used for *ChIP-seq* data.

The *filter_regions* operation requires (i) the *query_id* of the data that should be filtered; (ii) the name of the column on which filtering is applied; (iii) the comparison operation, `==` or `!=` for *string* type columns or `==`, `!=`, `>`, `>=`, `<`, or `<=` for numeric columns; (iv) the value to which the column value will be compared; (v) and the type of the aforementioned value (*number* or *string*). The command *filter_regions* is a versatile and useful operation for filtering specific (epi)genomic regions. Millions of regions can be selected in the first step of a workflow and a specific filtering can be applied for finding the regions that match its column content. Furthermore, it is possible to filter regions by multiple queries connecting the filtering operation by their output *query_id*. Listing 4.3 exemplifies the filtering operation in different columns.

```

1 # Selecting the data from 2 experiments, and the data in the chromosome 1.
2 (status, query_id) = server.select_experiments (["BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38
   .20150527.bed", "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"], "chr1", None,
   None, user_key )
3
4 # Filter the regions where the SIGNAL_VALUE is higher than 10
5 (status, query_id_filter_signal) = server.filter_regions (query_id, "SIGNAL_VALUE", ">", "10"
   , "number", user_key )
6
7 # Filter the regions where the PEAK value is higher than 1000
8 (status, query_id_filters) = server.filter_regions (query_id_filter_signal, "PEAK", ">", "
   1000", "number", user_key )

```

Listing 4.3: Example of the *filter_regions* operation: Line 2 selects the genomic regions from the experiments that are in the chromosome 1. After the regions that have the value of the column *SIGNAL_VALUE* higher than 10 are selected (Line 5), follows another filter (line 8). This last filter ensures that the value of the column *PEAK* is higher than 1000.

Meta-fields can be used by the *filter_regions* operation. For example, the code `(status, q_long_id)= server.filter_regions(data_id, "@LENGTH", ">=", 1000, "number", "anonymous_user")` selects the regions which are at least 1000 *bp* long. It is also possible to filter regions by their experiment metadata information: `(status, q_dnameth_id)=server.filter_regions(data_id, "@EPIGENETIC_MARK", "=", "DNA Methylation", "string", "anonymous_user")`. Another useful application is to filter regions by their CG content in the DNA sequence: `(status, q_cg_id)= server.filter_regions(data_id, "@COUNT.MOTIF(CG)", ">", "10", "number", "anonymous_user")`, which selects the regions that have at least ten CG in its DNA sequence.

Filtering by DNA sequence motif

The operation *filter_by_motif* allows users to filter for regions whose DNA sequence matches a specific regular expression pattern. In this operation, the *DeepBlue Server* accesses the DNA sequence of each region and verifies if its content matches the given regular expression. This operation is specially useful for finding regions of specific *TFBS* patterns or with CpG content. Listing 4.4 demonstrates the use of this operation.

```

1 # Filter the regions that DNA Sequence match the pattern [CG]+
2 (status, query_cg_motif) = server.filter_by_motif(query_id_filters, "[CG]+", user_key)
3
4 # Filter the regions that DNA Sequence match the pattern ACTAAAA
5 (status, query_acta_motif) = server.filter_by_motif(query_cg_motif, "ACTAAAA", user_key)

```

Listing 4.4: Example of the *filter_by_motif* operation: it is a continuation of Listing 4.3 source code after filtering by the regions column content. It filters the regions by their respective DNA sequence that must match the regular expression pattern *[CG]+*, followed by another filtering where the DNA sequence must matches the pattern *ACTAAAA*. Regions referenced by the *query ID query_acta_motif* overlap a DNA sequence containing sub-sequences that match both patterns.

Filtering by intersecting and overlapping

The *DeepBlue Server* provides the functionality of filtering regions by overlapping similar to *BEDTools* (Quinlan and Hall 2010), *WiggleTools* (Zerbino *et al.* 2013), and *BEDOPS* (Neph *et al.* 2012). The API provides two operations for this task: *intersection* and *overlap*.

The *intersection* operation is the simplest. It requires two *queries IDs*: one is referencing the data regions that contain the regions of interest and another is referencing the filtering region, which is used for filtering, for example, the CpG island annotation. The return value of this operation is a *query ID* that refers to the subset of the data regions that overlap with at least one of the filtering regions. Listing 4.5 presents an example, which uses the *find_motif* operation for generating a list of regions and use these regions for filtering region-sets from an epigenomic experiment²⁴.

```

1 # Find all locations where the motif TATAAA appears in the genome
2 (status, tataa_id) = server.find_motif("TATAAA", "GRCh38", "chr1", None, None, False,
    user_key)
3
4 # Selecting the data from 2 experiments: BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38.20150527.bed
    and S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed
5 # It selects the area in the chromosome 1, position 0 to 50.000.000.
6 (status, query_id) = server.select_experiments (["BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38
    .20150527.bed", "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"], "chr1", 0,
    50000000, user_key )
7
8 # Intersect the experiment regions with pattern
9 (status, intersected_id) = server.intersection(query_id, tataa_id, user_key)

```

Listing 4.5: Example of *find_motif* and *intersection* operations: it finds the regions in two epigenomic experiments that intersect with the pattern *TATAAA*. The variable *intersected_id* in line 9 refers to all *query_id* regions that overlapped to at least 1 bp of a *tataa_id* regions.

²⁴ It is a simple illustrative example and such a task can be replaced by the *filter_by_motif* operation

The *intersection* operation is useful, but usually a fine-tuning in the overlapping criteria is necessary, e.g. concerning the length of the overlap or performing an inverse filtering by removing the overlapping regions rather than keeping them. For cases where the *intersection* operation does not fulfill the user needs, the *overlap* operation is a more complex option for filtering regions regarding overlap criteria. This operation allows users to specify if the region *must* or *must-not* overlap, how much it must overlap, or how distant two regions must be in case of non-overlapping. This last constraint can be given in *bp* or in fraction of the region length.

The two first parameters of the *overlap* operation are the same as for the *intersection* operation (a pair of *query IDs*), but its third parameter specifies whether the regions must (*True*) or must-not (*False*) overlap. For example, the source code: `(res, qid_3)=server.overlap(qid_1, qid_2, False, 0, "bp", user_key)` removes all regions from the *qid_1* that overlap at least with one *qid_2* region.

The forth and fifth parameters for the *overlap* operation specify how many *bp* must overlap to be filtered. While the operation *intersection* accepts all overlaps, even if by just 1 *bp*, the *overlap* allows users to define how many *bp* must overlap. This option is demonstrated in Listing 4.6, where the overlap must be 6 *bp* long (line 9).

```

1 # Find all locations where the motif TATAAA appears in the genome
2 (status, tataa_id) = server.find_motif("TATAAA", "GRCh38", "chr1", None, None, False,
    user_key)
3
4 # Selecting the data from 2 experiments: BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38.20150527.bed
    and S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed
5 # It selects the area in the chromosome 1, position 0 to 50.000.000.
6 (status, query_id) = server.select_experiments (["BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38
    .20150527.bed", "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"], "chr1", 0,
    50000000, user_key )
7
8 # Overlap the experiment regions with pattern and the overlap must be at least 6 bp long
9 (status, intersected_id) = server.overlap(query_id, tataa_id, 6, "bp", user_key)

```

Listing 4.6: Example of *find_motif* and *overlap* operations: it finds the regions that fully overlap the pattern TATAAA.

The overlap width can be defined proportionally to the length of the regions. In this case, it is necessary to use '%' in the fifth parameter and specify the percentage in the fourth parameter. For example, the source code: `(res, qid_3)=server.overlap(qid_1, qid_2, True, 25, "%", user_key)`, specifies that at least 25% of the region in the *qid_1* must overlap with a region of the *qid_2*.

When using the operation *overlap* for filtering regions with no overlap, it is possible to specify the minimum distance between the regions. For example, the source code `(res, q_peaks_far CGI_id)=server.overlap(q_chip_peaks_id, q CGI_id, False, 1000, "bp", user_key)` filter all regions in the *q_chip_peaks_id* that are at least 1000 *bp* distant to any region in the *q CGI_id*. This example can be perceived as the regions in *q_chip_peaks_id* as *ChIP-seq* regions and *q CGI_id* regions as CpG islands, therefore, this code filters the *ChIP-seq* regions that are distant to any CpG island.

Furthermore, for defining the distance between non-overlapping regions, the *overlap* operation allows to use percentages proportional to the length of the regions rather than constant values. For example, the source code `(res, q_far_id)=server.overlap(qid_1, qid_2, False, 250, "%", user_key)` specifies that the *qid_1* regions must be at least 2,5 times its own length away from any *qid_2* region.

Region transformation

The *DeepBlue Server* API also allows to transform the region directly on the server, which enables extending the regions or generating flanking regions.

The *extend* operation allows for extending regions in the forward, backward, and both directions. This operation receives a *query_id*, the length of the expected extension, and the direction: *FORWARD*, *BACKWARD*, or *BOTH*. For example, the operation `(res, q_extended_both_id)=server.extend(regions_id, 25000, "BOTH", False, user_key)` extends the regions referenced by the variable *regions_id* by 25000 *bp* in both directions. The constant *BOTH* can be changed to *FORWARD*, for extending only from the ending position, or to *BACKWARD*, for extending from the starting position. This operation considers the region strand if the forth parameter is set as *True* and if this information is present in the region.

The *flank* operation generates flanking regions based on the existing regions. This operation requires a *query_id* that references the original region set, the starting position of each flanking region with respect to the original regions, the length of the flanking regions, and the information whether it must use the original regions *STRAND* column for defining the start and end of each region. The parameter *start* may have a negative value, for flanking before the region start, or a positive value, for flanking after the end of the region.

The promoter regions generation at run-time is one of the most convenient use cases for the *flank* operation. Listing 4.7 (line 20) exhibits an example where a list of genes is selected and their promoters are generated by the *DeepBlue Server*. In this example, the promoters start 2500 *bp* before the genes *Transcription Start Site* (TSS) and are 2000 *bp* long. In line 23 of the same example, another set of flanking regions are generated, starting 1500 *bp* after the gene body and have 500 *bp* length.

```

1 # Select genes by name
2 # For selecting all genes, set gene_names as None
3 gene_names = ["RNU6-1100P", "CICP7", "MRPL20", "ANKRD65", "HES2", "ACOT7", "HES3", "ICMT"]
4
5 # Select the gene from gencode v23
6 (status, q_genes) = server.select_genes(gene_names, None, "gencode v19", None, None, None,
    user_key)
7
8 # Generate flanking region that starts 2500 bp before the regions start and have 2000 bp.
9 # This region can be considered as promoter regions.
10 # The 4th argument ensures that DeepBlue must consider the region strand (column STRAND)
11 # to calculate the new region, so, the promoter region will be consist to the region start
12
13 (s, gene_promoters_id) = server.flank(q_genes, -2500, 2000, True, user_key)
14 (s, after_flank_id) = server.flank(q_genes, 1500, 500, True, user_key)
15
16 # Merge both flanking regions set and genes set
17 (s, all_merge_id) = server.merge_queries(q_genes, [gene_promoters_id, after_flank_id],
    user_key)

```

Listing 4.7: Example of *select_genes*, *flank*, and *merge* operations: it generates flanking regions before and after the genes, and merges all regions *query IDs*.

Merging queries

The *merge_queries* operation combines region sets of different *query_ids* into one *query_id*. This operation receives one *query ID* as the first parameter, and a list of *query IDs*. It

returns a *query ID* that references a region-set containing the regions of all merged *query IDs*. Listing 4.7 (line 26) exhibits an example where three *query IDs* are merged into a single *query ID*.

Summarizing region-sets

The *aggregate* operation aggregates the regions of a *query ID* using another *query ID* regions as boundaries. The aggregation results are accessed through the @AGG.* meta-fields listed in Table 4.5.5.

Listing 4.8 exemplifies the use of the *aggregate* operation: the first line selects the experiment to be aggregated. In this example, only one experiment is being selected, but any region set defined by a *query ID* can be used. The boundary regions are selected in line 4, where the CpG island annotation is selected. The *aggregate* operation is executed in line 7, summarizing the values of the column *VALUE*. The aggregation result is obtained through the *query ID* stored in the variable *agg_id*, which is used in line 10 for filtering the summarized regions that aggregate at least one region (@ACC.COUNT > 0). In this example, only the meta-field @AGG.COUNT is used but other meta-fields are available, for example: @AGG.MIN, @AGG.MAX, @AGG.SUM, @AGG.MEDIAN, @AGG.MEAN, @AGG.VAR, and @AGG.SD. All these meta-fields are explained in Table 4.5.5.

```

1 (status, experiments_id) = server.select_experiments (["GC_T14_10.CPG_methylation_calls.
   bs_call.GRCh38.20160531.wig"], "chr1", None, None, user_key )
2
3 # Select the CpG Islands annotation from GRCh38
4 (status, cpg_islands_id) = server.select_annotations("CpG Islands", "GRCh38", "chr1", None,
   None, user_key)
5
6 # Aggregate the regions using the column VALUE
7 (status, agg_id) = server.aggregate (experiments_id, cpg_islands_id, "VALUE", user_key )
8
9 # Select the summarized regions that aggregated at least one region
10 (status, flt_id) = server.filter_regions(agg_id, "@AGG.COUNT", ">", "0", "number", user_key)

```

Listing 4.8: Example of *aggregate* and *filter* operations: it aggregates the DNA methylation levels by CpG island regions.

Caching operation

In many cases, the data referenced by a *query ID* is re-used by different workflows. For improving the efficiency of the data retrieval and processing time, the *DeepBlue Server* API provides the *query_cache* operation, which caches the region set temporarily in the *DeepBlue Server* main memory.

The operation's cache stores up to 16 operation results²⁵ and follows the *Least recently used* (LRU) policy. The LMR policy defines that the least accessed cached result set is removed when a new result set must be stored in the cache.

The *query_cache* operation operation is a recommendation for the *DeepBlue Server* to cache the *query ID* result set, but there is no guarantee that the *query ID* result stored in the cache will be available when it be requested. This operation does not influence a workflow result besides improving the processing time.

²⁵ This number can be modified in the *DeepBlue Server* source code.

4.5.6 Data retrieval

The *DeepBlue Server* API provides different operations for retrieving a *query ID* region-set. These regions can be counted, enriched, retrieved as a score matrix or as a tabular file. In the following, the operations for such tasks are presented and explained. These operations do not return a *query ID* but a *request ID*, which represents a workflow processing request. The status of a *request ID* can be verified using the *info* operations, and its result is downloaded using the *get_request_data* operations. The *get_request_data* operation requires that the workflow processing is finished, so, it is necessary to verify the request status before executing it.

Listing 4.9 presents a useful function, `__wait_and_get_data()`, which waits for the workflow processing to finish and then returns the resulting data. This function is used in the following examples for downloading the workflow results.

```

1 # Wait for the server processing and return the data
2 def __wait_and_get_data(request_id, user_key):
3     (status, info) = server.info(request_id, user_key)
4     request_status = info[0]["state"]
5     while request_status != "done" and request_status != "failed":
6         time.sleep(1)
7         (status, info) = server.info(request_id, user_key)
8         request_status = info[0]["state"]
9
10    return server.get_request_data(request_id, user_key)

```

Listing 4.9: Example of *get_request_data* operation: it waits for a workflow processing finishes and then download its data. The function `__wait_and_get_data` is used in all following examples.

The following sections present the operations used for retrieving the workflow data.

Counting regions

The *count_regions* operation counts how many regions are referenced by a *query_id*. It is a straightforward operation, which receives a *query ID* and counts how many regions are referenced by it. Listing 4.10 exemplifies its usage.

```

1 # Selecting the data from 2 experiments
2 (status, query_id) = server.select_experiments (["BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38
   .20150527.bed", "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"], None, None, None,
   user_key )
3
4 # Count how many regions were selected
5 (status, request_id) = server.count_regions(query_id, user_key)
6
7 # Function for waiting and downloading the data
8 (status, count) = __wait_and_get_data(request_id, user_key)
9
10 print "The two experiments have", count["count"], "regions"

```

Listing 4.10: Example of *select_experiments* and *count_regions* operations: it shows how to count the number regions referenced by a *query ID*.

Calculating statistical values by binning

The *binning* operation provides a simple statistical analysis which groups the regions' column values into a group of "*bins*" and returns the count of items in each bin, working

as a generalization of a histogram. For example, using a DNA methylation experiment data, it is possible to create *bins* to obtain the distribution of the DNA methylation values among the different levels contained in the selected regions. The *binning* operation requires three parameters: a *query ID* with the regions to be binned, the name of the column whose values are binned, and the number of bins. Listing 4.11 presents a use case where the values of the column *SIGNAL_VALUE* of two experiments are binned in 20 bins.

```

1 # Selecting the data from 2 experiments, and the data in chromosome 1.
2 (status, query_id) = server.select_experiments(["BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38
   .20150527.bed", "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"], "chr1", None,
   None, user_key)
3
4 (status, request_id) = server.binning(query_id, "SIGNAL_VALUE", 20, user_key)
5
6 # Function for waiting and downloading the data
7 (status, bins) = __wait_and_get_data(request_id, user_key)
8
9 # Result: bins
10 # {'binning':
11 #   {'ranges': [2.3601, 5.513, 8.6658, 11.8187, 14.9716, 18.1245, 21.2774,
12 #              24.4303, 27.5832, 30.7361, 33.889, 37.0419, 40.1948, 43.3477,
13 #              46.5006, 49.6535, 52.8064, 55.9593, 59.1122, 62.2651, 65.418],
14 #   'counts': [5016, 3195, 1294, 716, 379, 252, 169, 104, 69, 48, 36, 21,
15 #              12, 6, 3, 3, 3, 0, 0, 0]}}
```

Listing 4.11: Example of the *binning* operation: it bins the PEAK_SIGNAL from *ChIP-seq* experiments in 20 bins.

Computing coverage

The *coverage* operation computes the coverage of the *query ID* regions in relation to a genome and its chromosomes. It merges all regions, calculates the total length, and returns a dictionary with the individual coverage (from 0.0 to 1.0) by chromosome, the chromosome total length, and the total coverage in *bp*. Listing 4.12 demonstrates the use of the *coverage* operation for computing the coverage of the CpG island in the genome *hg19*.

```

1 (res, cpq_id) = server.select_annotations("Cpg Islands", "hg19", None, None, None, user_key)
2
3 status, req = server.coverage(qid, "HG19", user_key)
4
5 status, coverage = __wait_and_get_data(req, user_key)
6
7 ## Result:
8 #{'coverages':
9 #  {'chr1': {'coverage': 0.7549, 'size': 249250621, 'total': 1881629},
10 #  'chr10': {'coverage': 0.7134, 'size': 135534747, 'total': 966963},
11 #  ## Removed for sake of space
12 #  'chrX': {'coverage': 0.4718, 'size': 155270560, 'total': 732552},
13 #  'chrY': {'coverage': 0.1951, 'size': 59373566, 'total': 115810}}}
```

Listing 4.12: Example of the *coverage* operation: it calculates the coverage of the annotation CpG island in relation to the genome assembly *hg19*.

Obtaining the distinct column values

The *distinct_column_values* operation obtains the distinct values for a given column. For example, Listing 4.13 demonstrates how to obtain the distinct states names from a CSS region-set.

```

1 status, csss_query_id = server.select_regions(None, "GRCh38", "Chromatin State Segmentation",
2       None, None, None, None, None, None, user_key)
3 (status, csss_names_request_id) = server.distinct_column_values(csss_query_id, "NAME",
4       user_key)
5 (status, state_names) = __wait_and_get_data(csss_names_request_id, user_key)
6
7 ## Output:
8 # {'distinct': {'10_Distal_Active_Promoter_2Kb_High': 2795055,
9 # '11_Active_TSS_High_Signal_H3K4me3_H3K4me1': 3606067,
10 # '12_Active_TSS_High_Signal_H3K4me3_H3K27Ac': 1995138,
11 # '1_Repressed_Polycomb_High': 1210683,
12 # '2_Repressed_Polycomb_Low': 4374048,
13 # '3_Low_signal': 12858129,
14 # '4_Heterochromatin_High': 4331726,
15 # '5_Transcription_High': 5671586,
16 # '6_Transcription_Low': 11313902,
17 # '7_Genic_Enhancer_High': 1825808,
18 # '8_Enhancer_High': 13427456,
19 # '9_Active_Enhancer_High': 3985090}}
```

Listing 4.13: Example of the *distinct_column_values* operation: it obtains the CSS names from a region-set.

Obtaining the experiments used in a region set

The *get_experiments_by_query* operation list the experiments that contain regions referenced by the given *query ID*. For example, a user selected data for a given epigenetic mark and then applied some filtering operations on this region set. Using the *get_experiments_by_query* operation is possible to obtain the experiments which have regions in the resulting filtered region-set.

Retrieving a region-set in tabular format

The *get_regions* operation outputs a column formatted file containing the region-sets referenced by the *query ID*. The output columns are configurable, where users can select the columns or use meta-fields (Section 4.5.5) for define the desired output.

This operation has two main parameters: the *query ID* that references the regions and the *format*. The *format* specifies the columns and meta-fields that must be included in the output. Listing 4.14 exhibits a complete workflow example: selecting, filtering, retrieving, and downloading a region set.

```

1  (status, samples) = server.list_samples("myeloid cell", {"source" : "BLUEPRINT genome"},
    user_key)
2
3  # Get the samples ID
4  samples_id = server.extract_ids(samples)[1]
5
6  # Select the regions from from chromosom 1, position 0 to 50.000
7  (status, query_id) = server.select_regions ("", "GRCh38", None, samples_id, None, None, "chr1",
    "", 0, 50000, user_key )
8
9  # Select the peak regions
10 (status, query_peaks_id) = server.query_experiment_type (query_id, "peaks", user_key )
11
12 # Retrieve the experiments data
13 # The @NAME meta-column is used to include the experiment name and @BIOSOURCE for experiment'
    s biosource
14 (status, request_id) = server.get_regions(query_id, "CHROMOSOME,START,END,@NAME,@SAMPLE_ID,
    @BIOSOURCE", user_key)
15
16 # Wait and download the data
17 (status, regions) = __wait_and_get_data(request_id, user_key)
18
19 print regions

```

Listing 4.14: Example of a workflow obtaining *ChIP-seq* data in a tabular format using the *select_regions* operation: this example lists, selects, and retrieves epigenomic data. First, it filters epigenomic experiments by their metadata: the *list_samples* operation obtains all samples of the biosource *myeloid cell* from the BLUEPRINT project, returning a list of samples with their *IDs* and description (line 9). The operation *extract_ids* extracts the *IDs* from this (line 12). These *IDs* are used it in the *select_regions* operation (line 15). The *select_regions* operation selects the genomic regions that are in the chromosome 1, position 0 to 50,000 in all experiments that have the selected samples *IDs*. Then, it uses the *get_regions* operation (line 21) with the parameters: *query_id* returned by the *select_regions* and the desired columns. The columns *@NAME*, *@SAMPLE_ID*, and *@BIOSOURCE* include the name, sample *ID*, and BioSource of the experiment in the output. The *get_regions* operation is asynchronous, so the user receives a *request_id* and should use the *info* operation to check the status of this request. Then, the method *__wait_and_get_data* facilitates the data download (line 24). Finally, the regions are printed to the user.

Retrieving a region set in a score matrix format

The *score_matrix* operation builds a matrix containing the aggregation result of the experiments data separated by the aggregation boundaries. This operation is similar to the *aggregate* operation, i.e., it requires a *query ID* with the regions' data to aggregate, and a *query ID* with the boundaries of the regions. However rather than returning a *query ID* it returns a *request ID* that references the constructed score matrix.

```

1 experiments = ["GC_T14_10.CPG_methylation_calls.bs_call.GRCh38.20160531.wig", "C003N351.
    CPG_methylation_calls.bs_call.GRCh38.20160531.wig", "C005VG51.CPG_methylation_calls.
    bs_call.GRCh38.20160531.wig", "S002R551.CPG_methylation_calls.bs_call.GRCh38.20160531.
    wig", "NBC_NC11_41.CPG_methylation_calls.bs_call.GRCh38.20160531.wig", "bmPCs-V156.
    CPG_methylation_calls.bs_call.GRCh38.20160531.wig", "S00BS451.CPG_methylation_calls.
    bs_call.GRCh38.20160531.wig", "S00D1DA1.CPG_methylation_calls.bs_call.GRCh38.20160531.
    wig", "S00D39A1.CPG_methylation_calls.bs_call.GRCh38.20160531.wig"]
2
3 experiments_columns = {}
4 for experiment_name in experiments:
5     experiments_columns[experiment_name] = "VALUE"
6
7 (status, cpgs) = server.select_annotations("Cpg Islands", "hg19", None, None, None, user_key)
8 (status, request_id) = server.score_matrix(experiments_columns, "mean", cpgs, user_key)
9 (status, score_matrix) = __wait_and_get_data(request_id, user_key)

```

Listing 4.15: Example of the *score_matrix* operation: it builds a score matrix containing the DNA methylation experiments summarized by the CpG island annotation.

4.5.7 Data enrichment

The *DeepBlue Server* can perform data enrichment analysis. It offers two types of enrichment analysis: by overlapping regions and by overlapping genes annotated by *Gene Ontology* (GO) terms. In the following, these methods are presented and also how they are implemented and optimized for the *DeepBlue Server* use cases.

Enrichment by overlapping regions

The *enrich_regions_overlap* operation performs enrichment analysis by processing the overlapping between region sets. This operation implements the *Locus Overlap Enrichment Analysis* (LOLA) (Sheffield and Bock 2016)²⁶ method directly on the server²⁷. The *DeepBlue Server* implementation has improvements on the original implementation, such as multi-threading and lower memory consumption.

Algorithm 4.1 presents the *LOLA* algorithm. It has three inputs: the *query* region-set, the *universe* region set, and a list of *datasets*. The *universe* is used to define which regions are basis of the analysis. For example, the *universe* can be composed of all promoter regions, or genes, but it can be also tiling regions for simpler analysis. The *datasets* are used for enriching the *query* regions. In the *DeepBlue Server*, the *query* regions and *universe* regions are represented by *query IDs*, which means that the user can perform a data selection, filtering, and subsequently, the enrichment directly on the server. The *datasets* input is a list containing experiment names or *query IDs*. Section 4.6.3 presents a complete use case of regions enrichment using the *enrich_regions_overlap* operation.

The *LOLA* algorithm, implemented in the *enrich_regions_overlap* operation processes the overlaps between the universe regions and query region to redefine the query region-set and then count the overlaps between the query and each dataset regions. With these values, a contingency matrix is build, and the Fisher exact test applied to it. The *p-value*, *natural log*, and *odds score* are calculated for each dataset. The datasets are ranked for each of these values and an average rank is calculated, totaling four results. Finally, the datasets are returned to the user with their ranking results.

²⁶ Source-code: <https://github.com/nsheff/LOLA/blob/master/R/calcLocEnrichment.R>

²⁷ Implementation file: *DeepBlue/server/src/processing/lola.cpp*

Algorithm 4.1: Locus Overlap Enrichment Analysis (LOLA) in the DeepBlue Server**Input:**

Q , query regions
 U , universe regions ▷ background regions
 DS , datasets

Output:

VE , a vector of combined ranking scores for each dataset

```

 $rQ \leftarrow \text{overlap}(U, Q)$  ▷ redefine  $Q$  to  $U$  regions that overlap to  $Q$  regions
for  $i \in \{1, \dots, DS\}$  do
   $support_i \leftarrow \text{countOverlaps}(rQ, DS_i)$ 
   $UoD_i \leftarrow \text{countOverlaps}(U, DS_i)$ 
   $b_i \leftarrow UoD_i - support_i$ 
   $c_i \leftarrow |rQ| - support_i$ 
   $d_i \leftarrow |U| - support_i - b_i - c_i$ 
   $pValue_i \leftarrow \text{fisherExactTest}(support_i, b_i, c_i, d_i)$ 
   $naturalLog_i \leftarrow -(\log_{10}(pValue_i))$ 
   $oddsScore_i \leftarrow \frac{a_i/b_i}{c_i/d_i}$ 
   $VE_i \leftarrow (DS_i, pValue_i, naturalLog_i, oddsScore_i)$  ▷ Store the result
end for

```

Fast enrichment analysis by regions bitmap

The *enrich_regions_overlap* performs reliable regions enrichment analysis, but it is not fast enough for interactive data analysis performed by tools like *DIVE* (Chapter 7). For this purpose, the *DeepBlue Server* provides the *enrich_regions_fast* operation. This operation uses an algorithm (Algorithm 4.2) similar to *LOLA*. But each region set, including the query, universe, and datasets, is converted to a bitmap that rather than comparing region sets through overlapping, the converted bitmaps are compared through an *and* bitwise operation. The overlaps count is calculated by counting how many bits are *true* in the bitwise operation result.

For improving its operation processing time when this operation compares a region set for the first time, the region set is processed, generating a bitmap, which is stored in the database. In this way, each region-set (experiment, annotation, *query ID*) bitmap is processed only once. The bitmap memory requirement is low: a bitmap with 2^{20} bits requires 128 kilobytes. Considering the overhead with the encapsulating BSON document and its content, the memory, and disk consumption per region-set bitmap is lower than 200 kilobytes.

A main difference between *enrich_regions_overlap* and *enrich_regions_fast*, is that the latter does not use the concept of *universe* and it does not receive a list of *datasets* to be enriched, but parameters for selecting the experiments to be used for the enrichment based on their metadata content.

The *enrich_regions_fast* operation also builds a contingency matrix for executing Fisher's exact test, returning the same mathematical values and ranks as the *LOLA* method. This operation does the same processing as *LOLA* if the *LOLA* universe parameter is defined as tiling regions of the same size as $genomeLength/2^{20}$.

Algorithm 4.2: Fast Locus Overlap Enrichment Analysis**Input:** Q , query regions DM , datasets selection metadata $BITMAP_SIZE$, size of the bitmap▷ DeepBlue uses 2^{20} **Output:** VE , a vector of combined ranking scores for each dataset

```

bmQ  $\leftarrow$  bitmap ( $Q, BITMAP\_SIZE$ )           ▷ build the query regions bitmap
cQ  $\leftarrow$  countTrue ( $DS_i$ )                   ▷ count the number of peaks
DS  $\leftarrow$  loadDataSets ( $DM$ )                 ▷ load datasets that match the given metadata
for  $i \in \{1, \dots, DS\}$  do
    bmDSi  $\leftarrow$  bitmap ( $DS_i, BITMAP\_SIZE$ )   ▷ build the dataset regions bitmap
    supporti  $\leftarrow$  bmQ  $\wedge$  bmDSi
    cDSi  $\leftarrow$  countTrue ( $DS_i$ )
    bi  $\leftarrow$  cDSi  $-$  supporti
    ci  $\leftarrow$  cQ  $-$  supporti
    di  $\leftarrow$  BITMAP\_SIZE  $-$  supporti  $-$  bi  $-$  ci
    pValuei  $\leftarrow$  fisherExactTest (supporti, bi, ci, di)
    naturalLogi  $\leftarrow$   $-(\log_{10}(pValue_i))$ 
    oddsScorei  $\leftarrow$   $\frac{a_i/b_i}{c_i/d_i}$ 
    VEi  $\leftarrow$  (DSi, pValuei, naturalLogi, oddsScorei)   ▷ Store the result
end for

```

The *enrich_regions_fast* operation compresses the region set in bitmaps of the same size (2^{20}). Bitmaps of the same size are compared in a $O(1)$ (constant time), while *LOLA* requires $O(n)$ (n being the number of regions) comparisons. The region sets compression procedure is straightforward:

1. Define bitmap size ($BITMAP_SIZE$). (Currently, it is 2^{20} .)
2. Divide the total genome size in *bp* by $BITMAP_SIZE$ and obtain the *tilingLength*
3. Generate tiling regions (*tiling_regions*) using the *tilingLength* value
4. For each region set (query and selected datasets):
 - a) Overlap the input region set with the tiling regions, marking the overlapped tiling regions
 - b) Iterate through the marked tiling regions and set the corresponding bitmap bit to *True*
 - c) Store the generated bitmap into the database

Finally, this operation is useful for fast and interactive enrichment, where the overlapping approximation results are enough for an overview of the data. A meaningful use case is when the user wants to perform an enrich analysis on the regions referenced by *query ID* against the available datasets. This case is performed by *DIVE* (Section 7.5.1) when searching for similar datasets.

Enrichment analysis by Gene Ontology

Besides the analysis of enrichment by overlapping regions, the *DeepBlue Server* analysis of GO-term enrichment through the *enrich_regions_go_terms* operation. This operation has only two main parameters: the *query ID* and the gene-model that is used to enrich the regions.

Algorithm 4.3 displays the *Gene Ontology Enrichment Analysis*. This algorithm starts with loading the gene regions and GO terms from these genes. After, overlaps between genes and *query* regions are processed, and the GO terms of the overlapping genes are obtained. In the next step, the algorithm iterates over the GO terms, calculating their *p-value* and *odds score* based on the count of overlaps.

Algorithm 4.3: Gene Ontology Enrichment Analysis

Input:

Q , query regions
 GM , gene model

Output:

VE , a vector of combined ranking scores for each gene ontology term

```

AllGenes  $\leftarrow$  LoadGenes ( $GM$ )
AllGoTerms  $\leftarrow$  GetGOTerms ( $AllGenes$ )
OverlappedGenes  $\leftarrow$  overlap ( $AllGenes, Q$ )
OverlappedGoTerms  $\leftarrow$  GetGOTerms ( $OverlappedGenes$ )
for  $i \in \{1, \dots, OverlappedGoTerms\}$  do
    support $i$   $\leftarrow$  OverlapCount ( $OverlappedGoTerms_i$ )
    b $i$   $\leftarrow$  TotalCount ( $OverlappedGoTerms_i$ ) - support $i$ 
    c $i$   $\leftarrow$  | $OverlappedGenes$ | - support $i$ 
    d $i$   $\leftarrow$  | $AllGenes$ | - support $i$  - b $i$  - c $i$ 
    pValue $i$   $\leftarrow$  fisherExactTest (support $i$ , b $i$ , c $i$ , d $i$ )
    naturalLog $i$   $\leftarrow$  -( $\log_{10}(pValue_i)$ )
    oddsScore $i$   $\leftarrow$   $\frac{a_i/b_i}{c_i/d_i}$ 
    VE $i$   $\leftarrow$  ( $DS_i, pValue_i, naturalLog_i, oddsScore_i$ ) ▷ Store the result
end for

```

A much simplified analysis can be performed with the *count_gene_ontology_terms* operation, which returns how many times a GO term appears in the given data selection.

4.5.8 Data and metadata insertion and maintenance

The *DeepBlue Server* API provides a set of operations for inserting and maintaining its metadata entities content. These operations are executed only by the administrator or by authorized users. Table 4.4 presents an overview of these operations, Appendix A.4 provides all available operations.

Among these operations, the *clone_dataset* operation provides a facilitated way for curating the (epi)genomic data stored in the *DeepBlue Server*. This operation duplicates a dataset (experiment or annotation) allowing the user to specify new metadata content

for the cloned dataset, without losing the old dataset information. Furthermore, the existing regions are not duplicated, but the new dataset references the existing data.

Category	Operation	Description
Vocabularies terms	<i>add_epigenetic_mark</i>	Include new Epigenetic Mark
	<i>add_biosource</i>	Include new BioSource
	<i>add_sample</i>	Include new Sample
	<i>add_genome</i>	Include new Genome
	<i>add_technique</i>	Include new Technique
	<i>add_project</i>	Include new Project
(epi)genomic data	<i>add_annotation</i>	Include a new Annotation
	<i>add_experiment</i>	Include a new Experiment
Genes and GO	<i>add_gene_model</i>	Include a new Gene Model
	<i>add_gene_ontology_term</i>	Include a new Gene Ontology Term
	<i>annotate_gene</i>	Annotate a gene with a GO Term
Column types	<i>create_column_type_simple</i>	New simple column
	<i>create_column_type_category</i>	New categorical column
	<i>create_column_type_range</i>	New range column
	<i>create_column_type_calculated</i>	New calculated column
Experiments	<i>change_extra_metadata</i>	Change an experiment metadata
	<i>clone_dataset</i>	Clone a dataset
Remove	<i>remove</i>	Remove a DeepBlue Server entity

Table 4.4: DeepBlue Server operations to include and maintain metadata and data.

4.6 Usage examples

To illustrate the wide range of applications in which the *DeepBlue Server* can be used for the efficient retrieval of epigenomic data, this section presents five typical usage examples that had to be performed manually in epigenetic studies until now: (i) identification of *TFBSs* that overlap with *H3K4me3* peaks and promoter regions; (ii) calculating DNA methylation levels across *H3K4me3* peak regions; (iii) enriching *Differentially Methylated Regions (DMRs)* by Chromatin States; (iv) obtaining gene-specific (epi)genomic information from different tissues; (v) summarizing gene expression measurements from hepatocyte experiments. This section presents the central concepts, description, and results of the usage examples, while Appendix A.3 contains their source code with a comprehensive description. Furthermore, these usage examples are illustrative cases of how the *DeepBlue Server* can be used, and they be easily extended to different biological questions.

4.6.1 Identification of *TFBSs* that overlap *H3K4me3* peaks and promoter regions

This usage example aims to answer which TFs are active by looking at the *H3K4me3* peaks. The location of the *TFBSs* is known, but it is also necessary to verify if they are accessible. For this task, this usage case selects the TFs that overlap with *H3K4me3* which is known to signal accessibility and activity in promoter regions (Koch *et al.* 2007).

The selected regions are filtered for promoter regions in order to remove possible false positives.

This example demonstrates how the *DeepBlue Server* can operate on several data sources in the same workflow. First, the regions from 306 BLUEPRINT datasets annotated with *H3K4me3* peaks are selected. Afterwards, these regions are filtered by overlapping with promoter regions. The resulting regions are filtered again, this time, by overlapping with *TF* binding sites of *SP1* from the 44 ENCODE datasets. Finally, the columns of interest are selected, and the resulting regions are downloaded. Figure 4.17 shows the workflow diagram of this usage example and Appendix A.3.1 presents Listing A.2 with the complete source code of this example.

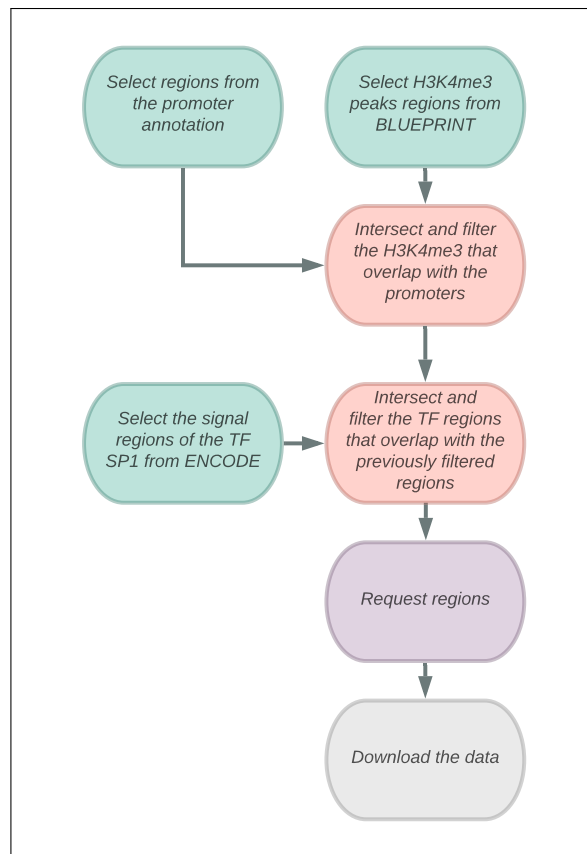


Figure 4.17: Workflow diagram for the identification of 3Kme3 peaks that overlap with promoters in any of the BLUEPRINT datasets and subsequent identification of TFBS peaks that overlap with these promoters in any of the ENCODE datasets. The different colors represent different types of operation, i.e. green for data selection, red for data manipulation, purple for processing requests, and gray for download.

This usage example demonstrates how the *DeepBlue Server* facilitates collecting large-scale data, in which all the *H3K4me3* peaks data from the BLUEPRINT Epigenome project were accessed and compared to dynamically generated promoter regions, as well as to data from another epigenomic project. This was achieved in approximately 40 lines of code and total runtime of 33 seconds. This usage example returns to the user a list of TFBSs ready to be operated in a data analysis environment.

4.6.2 Calculating DNA methylation levels across *H3K4me3* peak regions

This usage example demonstrates how to obtain summarized data from different BioSources and epigenetic marks, summarizing DNA methylation data from liver tissues in terms of *H3K4me3* peaks derived from *embryonic stem cells* (ESCs). The *H3K4me3* histone mark was selected because it acts as a marker for transcriptionally active promoters (Koch *et al.* 2007), whereas DNA methylation serves as a repressive mark when located in the gene promoters (Weber *et al.* 2007).

This usage example exemplifies how to obtain and summarize DNA methylation data using the *DeepBlue Server*. In this usage example, liver and hepatocyte samples, selected for illustrative purpose, are aggregated by *H3K4me3* peaks, where each generated data file contains the summarized regions of one sample.

Figure 4.18 shows the processing flow of this usage example: (i) list and select the regions from *H3K4me3* and ESCs experiments, for later use as summarizing boundaries; (ii) list all liver and hepatocyte experiments data file; (iii) for each experiment data file, aggregate its DNA methylation signal using the previously obtained *H3K4me3* peaks, download the results from the server, and store the summarized data in file.

The usage example source code is provided in Appendix A.3.1 (Listing A.3) together, with a full explanation of the source code. For potential future usages, parameters such as BioSources and epigenetic marks can be easily modified for obtaining different data. A future application of the data obtained in this usage example is correlating this data with gene expression data, or even building a gene expression predictor based on DNA methylation data.

4.6.3 Enrichment Analysis of DMRs regarding Chromatin States

DMRs are genomic regions with different DNA methylation levels across different samples, which result in a putative change of the transcription regulation genes. DMRs can either be hypomethylated, i.e. the DNA methylation level is lower than the methylation DMRs in a reference condition, or hypermethylated, meaning a higher DNA methylation level.

In this example, an enrichment analysis is performed with the goal of obtaining new insights on the sets consisting of DMRs, such as investigating which chromatin states are found in hypomethylated DMR. The *DeepBlue Server* API provides the *enrich_regions_overlap* operation that performs enrichment analysis similar to the LOLA (Sheffield and Bock 2016) method. The *enrich_regions_overlap* is detailed in the Chapter 4.5.7.

For meeting the goal of this usage example, it is also demonstrated how to dynamically generate CSS states region sets using the existing CSS data files. Listing A.4 in Appendix A.3.1 contains the usage example source code.

This usage example is divided into two parts: (i) extraction of CSS states regions from the experiments and construction of a region set containing the regions of each CSS state; (ii) selecting, filtering and enriching the region set. The extraction of the CSS states region is necessary because each CSS experiment contains all states, but the enrichment methods requires that each region set contains only one CSS state. Therefore, it is necessary to extract the region states of each experiment and group them in new region-sets. This task generates a set of tuples, each one contains an experiment name, a CSS state

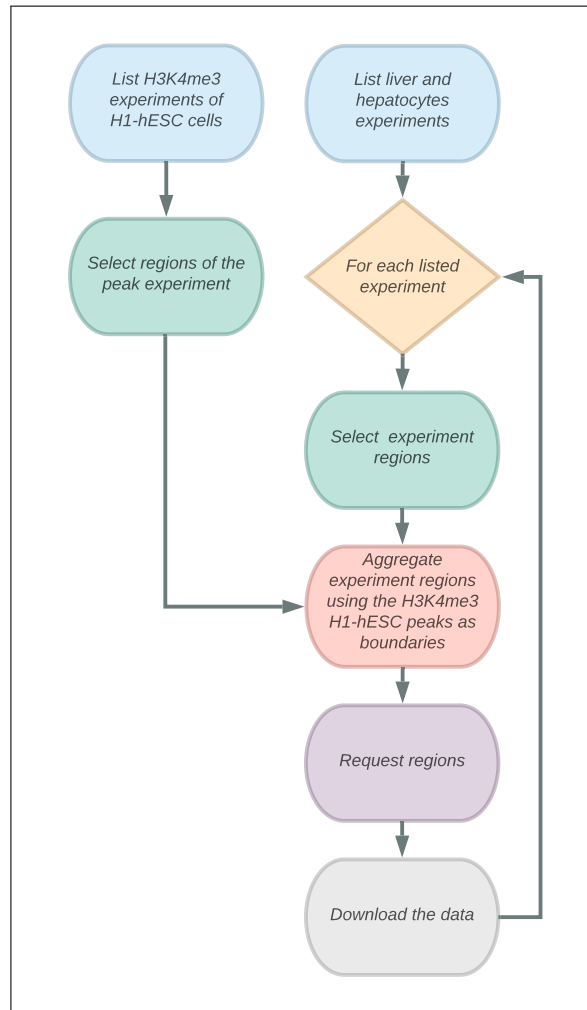


Figure 4.18: Workflow diagram for summarizing DNA methylation levels in liver tissue across *H3K4me3* peaks regions derived from human embryonic stem cells. The different colors represent different types of operation, i.e. green for data selection, red for data manipulation, purple for processing requests, and gray for download.

name, and the region-set extracted from this experiment and state. The process of generating the CSS states for the genome *GRCh38* can be performed dynamically (function *build_chromatin_state_files* in Listing A.4) by the *DeepBlue Server*. The following steps summarize this process:

1. Select all CSS experiment files from the *GRCh38* genome.
2. Obtain all CSS states contained in selected files. (Function *get_chromatin_states* in Listing A.4).
3. For each file:
 - a) For each CSS state, filter the CSS file regions by this state obtaining a *query ID* related to the experiment and state filtering. (Function *split_file* in Listing A.4).
4. Create a dictionary containing the CSS experiment file names as keys and a list of *query IDs*. Each *query ID* is associated to the regions of a CSS state in the file.

The enrichment analysis is then performed as follows:

1. Select the *DMR* file to be analyzed. For illustrative purpose, this usage example uses the file *S00VEQA1.hypo_meth.bs_call.GRCh38.20150707.bed* from the BLUEPRINT epigenome project, but any *DMR* data file can be used.
2. Filter the *DMR* file regions which the values of the column *AVG_METHYL_LEVEL* is lower than 0.0025.
3. Execute the enrichment operation *enrich_regions_overlap* using the filtered regions and dictionary defined .

The last lines of Listing A.4 sort the enrichment result by the most highly enriched states and print the results. Due to the output size, the raw results are omitted here²⁸. The most enriched state is the *12_Active_TSS_High_Signal_H3K4me3_H3K27Ac* (in the top 20 states), followed by *11_Active_TSS_High_Signal_H3K4me3_H3K4me1*, and the lowest ranked state is *4_Heterochromatin_High*. These findings are consistent with the general consensus of the gene regulation process, where CpG islands of actively transcribed genes are usually largely unmethylated to make them accessible to transcription factors (Lokk *et al.* 2014).

4.6.4 Obtaining gene-specific (epi)genomic information from different tissues

This usage example explores the expression difference of the *fatty acyl-CoA reductase 1* protein by the *FAR1* gene in the liver and brain tissues. The goal of this example is to demonstrate the simplicity of obtaining the epigenomic modification that may play a role in the expression of a specific gene.

Far1, a peroxisomal C-tail anchored protein, is the enzyme responsible for the synthesis of long chain alcohols. The Far1 protein regulates the biosynthesis of *ethanolamine plasmalogen* (PlsEtn), which is found in several organs, such as the brain, kidney, heart, and liver. In heart, PlsEtn constitutes approximately 50% of Etn-containing phospholipids, whereas in the liver, plasmalogens are only slightly detectable (Honsho and Fujiki 2017). For this reason it is assumed that the *FAR1* gene has a higher expression in the brain than in the liver. Based on this assumption, this usage example obtains the (epi)genomic data related to the *FAR1* gene and surrounding regions that are defined below.

In this usage example, data from the ROADMAP Epigenomic Project of tissues related to the brain (*Neurosphere Cultured Cells Cortex Derived*, *Neurosphere Cultured Cells Ganglionic Eminence Derived*, *Brain Germinal Matrix*, *Brain Hippocampus Middle*, *Fetal Brain Female*) and to the liver (*Adult Liver*, *HepG2 Hepatocellular Carcinoma*) are obtained. From these tissues, gene expression, gene accessibility (DNaseI), DNA methylation (RRBS and WGBS), and histone modification (*H3K36me3*, *H3K4me3*, *H3K27me3*, *H3K4me1*, *H3K9me3*, *H3K27ac*) data of the gene body and surrounding regions are obtained. The surrounding regions here refer to the promoter regions (starting 2,500 bp before the TSS with the length 2,000 bp) and after the gene body (starting 1,500 bp after the gene body with the length of 500 bp).

²⁸ The example source code with the output is available at <https://github.com/MPIIComputationalEpigenetics/DeepBlue/blob/master/examples/Enrichment%20by%20Chromatin%20States.ipynb>

The usage example and its source code, presented in Appendix A.3.1, can be divided into five sections: (i) loading packages and defining the generic function for downloading data from the *DeepBlue Server*; (ii) loading the *FAR1* gene location and determining the surrounding regions; (iii) defining the data of interest; (iv) obtaining the samples and experiments of the defined biological sources; (v) obtaining the data from the experiments, summarizing them by the gene body and surrounding regions, and writing the results in a file, where each file contains the data of an epigenetic mark.

Despite the limited number of samples used, some interesting insights can be gained (Table 4.5). First, indeed, the *FAR1* gene is expressed less highly in the liver than in the brain, but the difference is not very large. J. B. Cheng and Russell 2004 show a higher difference of the *FAR1* gene between brain and liver in different mammals²⁹. This difference may have been caused by the fact that the publication uses tissues from different animals and averages the results, whereas our usage example is based on data from a single human sample. It is important to note that the cell line *HepG2* expresses this gene to a much smaller extent than tissue obtained directly from the liver. The DNA methylation is higher in the promoters of the liver cells, probably acting as a gene-expression repressor. Other repressive marks, such as *H3K27me3* and *H3K9me3* are also more prevalent in the liver samples, with a larger difference in *H3K27me3*. Observing exclusively the *HepG2* cell line and brain samples, the values in the *H3K9me3* data do not show a clear distinction between these two groups. When observing the histone marks for active gene expression *H3K4me3*, *H3K27ac*, *H3K4me1*, there is a observable difference between the liver and brain cells but the marks *H3K9me3* and *H3K36me3* do not show a noticeable difference.

Due to space restrictions in the Table 4.5, the sample sources are referenced by their ROADMAP ID. For better visual recognition, the ROADMAP IDs are colored following the source of the sample, as shown in Table 4.6.

To summarize, this usage example shows how to investigate differences in the expression and regulation of the *FAR1* gene between the liver and brain samples. With the help of approximately 100 lines of code, it was possible to obtain the available data from the ROADMAP Epigenome project and verify the original biological assumption. This example can be easily modified for different genes, epigenetic marks, samples, and projects.

4.6.5 Summarize gene expression from hepatocyte experiments

This usage example demonstrates how to obtain and combine gene expression data from multiple experiments. It illustrates how to make the data collected from the *DeepBlue Server* available for analysis with the library *NumPy*³⁰ and to plot charts on this data using the *matplotlib*³¹. Listing A.6 in Appendix A.3.1 presents this usage example source code.

A set of genes highly expressed in the hepatocyte cells (*ADH1A*, *ADH1C*, *ADH4*, *ADH5*, *ADH6*, *ADH7*, *GSTA1*, *GSTA2*, *GSTA3*, *GSTA4*) is used by this example. The

²⁹ J. B. Cheng and Russell 2004 shows that *FAR1* gene has the expression level 63 times higher in the analyzed brain samples than in the analyzed liver samples.

³⁰ <http://www.numpy.org/>

³¹ <https://matplotlib.org/>

	Location	E053	E054	E066	E070	E071	E082	E118
RNA	Promoter							0.06
	Gene Body	2.11	2.58	1.69	3.32	5.59	2.07	0.17
	After gene	0.15	0.20			0.42	0.39	
DNase	Promoter	-	-	-	-	-	10.56	0.95
	Gene Body	-	-	-	-	-	4.370	1.28
	After gene	-	-	-	-	-	1.170	0.57
DNA methylation (WGBS)	Promoter	0.25	0.24	0.40	0.22	0.18	-	-
	Gene Body	0.78	0.72	0.73	0.73	0.73	-	-
	After gene	0.95	0.98	0.95	0.97	0.95	-	-
H3K27me3	Promoter	0.59	0.33	3.69	1.64	0.19	0.36	0.52
	Gene body	0.10	0.16	1.51	0.90	0.12	0.11	0.32
	After gene	0.46	0.14	0.96	0.64	0.003	0.16	0.89
H3K4me3	Promoter	90.01	68.37	6.00	52.02	55.08	86.88	0.28
	Gene body	39.57	38.39	0.94	19.50	12.34	35.63	0.17
	After gene	0.26	0.16	0.32	0.14	0.07	0.18	0.42
H3K27ac	Promoter	-	-	4.84	-	22.63	-	0.18
	Gene body	-	-	1.54	-	5.61	-	0.16
	After gene	-	-	0.26	-	0.10	-	0.23
H3K4me1	Promoter	4.18	1.28	0.92	1.29	8.15	1.23	0.15
	Gene body	1.49	0.83	0.36	0.41	2.90	0.67	0.13
	After gene	0.64	0.16	0.39	0.16	0.05	0.39	0.18
H3K9me3	Promoter	0.39	0.26	1.37	0.52	0.40	0.21	0.31
	Gene body	0.33	0.24	0.53	0.38	0.31	0.25	0.17
	After gene	0.59	0.11	0.29	0.59	0.10	0.14	0.17
H3K36me3	Promoter	0.19	0.39	0.26	0.24	0.43	0.13	0.20
	Gene Body	1.09	1.17	0.97	0.70	1.59	1.16	0.20
	After gene	0.91	0.52	0.28	0.57	0.29	0.44	0.29

Table 4.5: (Epi)genomic data obtained about the *FAR1* gene.

RNA-seq data of these genes are obtained from eight hepatocyte experiment files³² from the *AMED-CREST* project. The gene expression values are computed using the *score_matrix* operation, where the inputs are the experiment names and genes' location obtained using the *get_regions* operation.

The computed score matrix containing the gene expression values is used as input for the *NumPy* library, which constructs an array with facilitators for manipulating the score matrix data. This array serves as input for the *matplotlib* that draws the box plot displayed in Figure 4.19, showing the genes and their respective expression values.

This example demonstrates how the *DeepBlue Server* can empower users by facilitating the development of data visualization scripts that present summarized (epi)genomic data. Furthermore, this example can be easily modified for different gene sets or experiment data.

4.7 Discussion

Large volumes of epigenomic data are being generated, for instance, by the various *IHEC* members and reprocessing projects. These data hold the promise of revolutionizing our understanding of cell regulation and human diseases. However, studies aimed at fulfilling this promise are faced with the complexity of data acquisition and processing.

³² Section 6.3.1 contains the *DeepBlue Web Portal* usage example demonstrating this data selection.

Sample Source	ROADMAP ID	Source
Neurosphere Cultured Cells Cortex Derived	E053	Brain
Neurosphere Cultured Cells Ganglionic Eminence	E054	Brain
Adult Liver	E066	Liver
Brain Germina	E070	Brain
Brain Hippocampus	E071	Brain
Fetal Brain Female	E082	Brain
HepG2	E118	Liver

Table 4.6: ROADMAP IDs used for obtaining data on *FAR1* gene with the colors that are used in the following table.

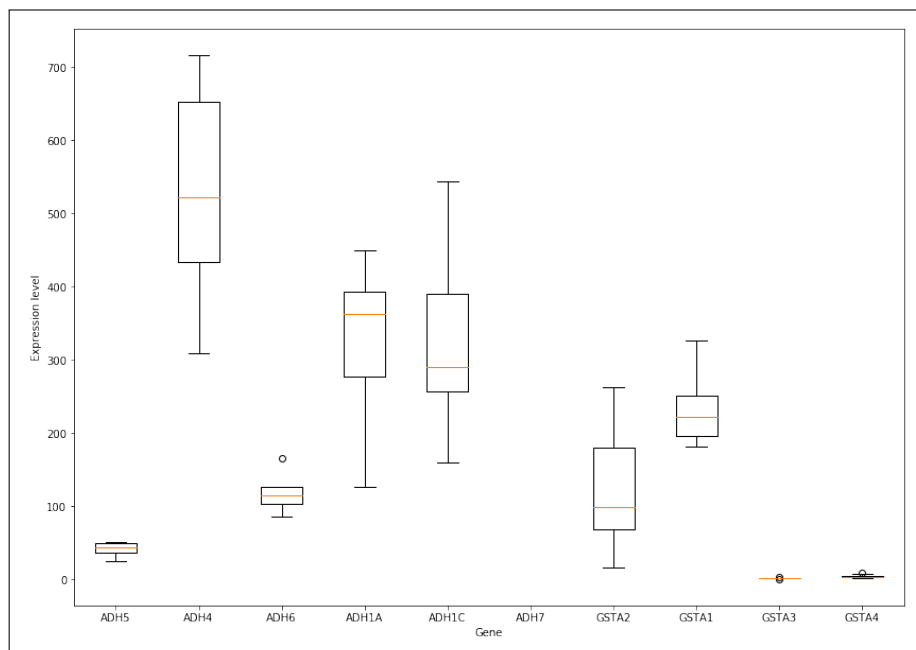


Figure 4.19: Summarize gene expression from hepatocyte experiments.

These steps are time-consuming due to the lack of suitable programmatic access. Currently, epigenomic data is only accessible through web portals, which are set up by the respective consortia. These portals offer access to region-set data and metadata but lack effective mechanisms for searching, filtering and processing of these data programmatically. Moreover, a lack of metadata standardization complicates the use of data from different sources in a single study. In a typical epigenomics analysis, such routine pre-processing steps are thus often more time-consuming than the following analyses and consequently hinder effective research in the field. Furthermore, researchers have access to terabytes of epigenomic data. This creates a strong demand for data processing and analysis, which is not effective on local computers due to the volume of the data.

The *DeepBlue Epigenomic Data Server* was developed to mitigate the problems described above. It is a data server that enables users of epigenomic data to find, select, manipulate, enrich, summarize, and retrieve region-set data from several epigenomic consortia, namely BLUEPRINT, DEEP, ENCODE, and REMC. The *DeepBlue Server* hosts peaks files, signal files, with their corresponding metadata, currently amounting to

64,113 experiments data files from 6,398 samples, and 100 annotations of various types of genomic regions.

The *DeepBlue Server* programmatic access through its API allows selecting and manipulating epigenomic data efficiently, making DeepBlue highly effective regarding answering specific research questions with comparably little effort. Users can combine simple data operations into complex workflows for automated retrieval of aggregated results relevant for the research question at hand. DeepBlue does not only streamline access to important research data but also fosters reproducibility, recording of the operations executed in each workflow.

Access to DeepBlue is possible anonymously. Nevertheless, the software has close to 160 registered users, mainly from Germany and European countries, but also from USA, Canada, Egypt, Israel, India, New Zealand, and Australia, who benefit from additional features such as access to private data, more computational resources, and a history of their activity. In addition to individual users, it is observed that DeepBlue is getting traction as a resource for organizing and retrieving data in other tools such as DEEP-Tools (Ramírez *et al.* 2016) and the BLUEPRINT Data Analysis Portal (J. M. Fernández *et al.* 2016).

DeepBlue has been under active development and extensive testing since 2014. It has been openly available for users outside our institute since September 2015, initially with 30 gigabytes of (epi)genomic data, increasing to 27 terabytes at present. Since that date, The *DeepBlue Server* has processed more than 3,570,000 workflow processing requests. From these requests, approximately 3,250,000 were made by anonymous users and the remaining by registered users.

Due to the large amount of stored and handled (epi)genomic data, optimizations were made in the *DeepBlue Server* in four main respects: (i) compressing the region sets data; (ii) indexing the regions for fast retrieval; (iii) using efficient algorithms that can handle large data volumes; (iv) implementing the data operations using parallel computing code. As a result of these software level optimizations, requests that involve the aggregation of millions of region-sets can typically be computed in a few minutes. The software also makes use of scalable solutions to cope with the increasing amount of data, such as (i) data processing: multiple instances of the *DeepBlue Server* on different servers can be started when the processing load is high; (ii) data storage: the MongoDB database facilitates the inclusion of new computer nodes in the database clusters to increase capacity.

The *DeepBlue Server* also addresses software architecture challenges, such as (i) scalability: the system needs to be able to cope with the currently available data as well as with the growing volume of epigenetic data that will be generated in the upcoming years; (ii) metadata standardization: it is imperative to handle all data accessible in a standardized form to increase the efficiency of future epigenomic data analysis and software development; (iii) usability: DeepBlue provides a simple API that users can access to operate on epigenomic data using any of a number of common programming languages; (iv) cloud computing: its architecture aligns with the current *cloud computing* paradigm, where it can be installed in a *cloud environment* and *DeepBlue Server* instances can be dynamically started when it receives workflow processing requests; (v) compatibility: access is provided through the XML-RPC or *RESTful* protocols, maximizing the compatibility with various programming languages.

For optimal user support, DeepBlue provides a user manual, API reference guide, step by step tutorials, code examples, and practical use cases. This allows users to quickly familiarize themselves with the relevant parts of our API. DeepBlue also provides a user-friendly and interactive web portal (Chapter 6), which serves as an example for a software application that utilizes the *DeepBlue Server* API efficiently and, in addition, enables users without programming expertise to search, browse, and download epigenomic data.

DeepBlue serves as a comprehensive online resource for the epigenomic community. It is unique in its ability to handle epigenomic data from different consortia in a single workflow. This is particularly advantageous for large-scale data analysis involving many different BioSources. However, there are limitations to this type of analysis caused by the differences in the raw data processing pipelines. Such pipelines differ across and even within the various consortia. For instance, using a different set of tools or even the same tools with different versions or parameters can have a significant impact on the results and lead to batch effects. Similarly, the use of data from different reference genome assemblies impacts on the exact location of the regions, introducing bias in the results. While this is an issue that cannot be immediately addressed, it is expected that the current efforts from the *IHEC* consortium members achieve a higher degree of data processing standardization, making multi-projects epigenomic data analyses more robust.

The development of DeepBlue was motivated by the author's involvement in the *DEEP* and *BLUEPRINT* projects, which also allowed him to closely communicate with the epigenomics community. As a result, *DeepBlue* has already received substantial interest from several members of the *IHEC* community. DeepBlue has the potential of finding widespread adoption as a tool for epigenomic data retrieval and processing both in software and in analysis pipelines used in future studies involving epigenomic data. Finally, DeepBlue has a stable API and implementation built on five years of experience in handling epigenomic data. With the features currently offered, DeepBlue has the potential of driving future computational epigenomic research.



5

DeepBlue in R/Bioconductor



The DeepBlueR Package is joint work with Dr. Markus List. The use example in Section 5.3.3 was conceptualized by Dr. Christoph Bock. The metadata cleanup and batch effect analysis (Section 5.3.4) was developed by Fawaz Dabbaghieh under the supervision of Markus List and the author. The DeepBlue R Package was published in Albrecht et al. 2017.

The *R/Bioconductor* environment is particularly popular for data analyses. The package *DeepBlueR*¹ was developed to streamline access to the *DeepBlue Server* API through features like data compression, caching, and transparent data conversion from the *DeepBlue Server* into *R* data structures.

This chapter presents the *DeepBlueR* package, starting by an overview in Section 5.1, examples in Section 5.2, use cases in Section 5.3, and conclusion in Section 5.4. Appendix A.2.3 presents the *DeepBlueR* installation instruction and implementation details.

5.1 Overview

The *DeepBlue Server* provides a powerful API for handling (epi)genomic data. The *R/Bioconductor* environment (R Core Team 2013; Gentleman, V. J. Carey, Bates, Bolstad, Detting, Dudoit, Ellis, Gautier, Ge, Gentry, et al. 2004b) is a popular environment for performing (epi)genomic data analysis. Even though it is possible to access the *DeepBlue*

¹ <https://bioconductor.org/packages/release/bioc/html/DeepBlueR.html>

Server from virtually all programming language, its access from the *R/Bioconductor* environment was not optimal. The reason resided in the *R XML-RPC* protocol² implementation which was not efficient and not fully compatible with the *XML-RPC* specifications (Winer 1999). For solving this issue, the *DeepBlueR* package was developed to streamline the (epi)genomic data processing workflow, starting from the data retrieval from the *DeepBlue Server* to downstream analysis performed in the *R/Bioconductor* environment. It enables users to quickly gather and transform epigenomic data from selected experiments for analysis in the *Bioconductor* ecosystem. The *DeepBlueR* package is included in the *Bioconductor* package repository, simplifying its installation.

DeepBlueR users combine commands in custom workflows for operating on the epigenomic data in the *DeepBlue Server*. With the exception of the *Data and Metadata insertion and maintenance commands* (sub-section 4.5.8), all operations available in the *DeepBlue Server* are available in *DeepBlueR*. Moreover, *DeepBlueR* provides new commands for efficiently downloading the data from the server and for saving the data into files. *DeepBlueR* has been optimized for speed, which includes an optimized *XML-RPC* protocol implementation, data compression, local caching of results, and converting the data to a different genome assembly (*liftover*). One of the most import features is the transparent conversion of the data retrieved from *DeepBlue* into suitable *R* data structures, such as *GenomicRanges* (Lawrence *et al.* 2013).

The integration of the *DeepBlue* into the *R/Bioconductor* environment adds access to many useful features to the *DeeBlue* users. The following list presents some examples.

Data visualization : *GViz* (Hahne *et al.* 2012; Hahne and Ivanek 2016) is a library for visualizing genomic and regulatory annotations. The *ggplot2* (Wickham 2009) is a powerful charting library widely used.

Annotation : *AnnotationHub* is a library that provides access to hundreds of different genomic annotations.

Integrative analysis : *TCGAbiolinks* (Colaprico *et al.* 2015) is a library for accessing *The Cancer Genome Atlas* (TCGA) data and for performing integrative analysis on this data.

Data manipulation : *matrixStats* is a library performing functions on rows and columns of matrices.

Machine learning : *glmnet* (J. Friedman *et al.* 2010; N. Simon *et al.* 2011) is library for fitting the entire lasso or elastic-net regularization path for linear regression, logistic and multinomial regression models.

5.2 Usage examples

DeepBlueR provides all data searching, listing, selecting, manipulating, sumarizing, enriching, and downloading operations provided by the *DeepBlue Server* API. It has three small differences: (i) the incorporation of the prefix *deepblue_* in the operation names; (ii) all optional parameters are assigned the value *NULL*, 0, or an empty string, by default; (iii) operation parameters can be accessed by their names. The *DeepBlueR* package also comprises other convenient features, such as an options handler with pre-configured values for the *DeepBlue Server URL* or the anonymous *user key*. It is possible to change these parameters using the command `deepblue_options(user_key="my_user_key")`.

² The package XMLRPC was deprecated and removed from *Bioconductor*

DeepBlueR provides full integration with *R* data structures and commands. As an example, Listing 5.1 presents the use of the *full-text search* command, followed by the *info* command, and extraction of the required metadata.

```

1 # Select the experiments with terms 'H3k27AC', 'blood', and
2 # 'peak' in the metadata.
3 experiments_found = deepblue_search(
4   keyword="'H3k27AC' 'blood' 'peak'", type="experiments")
5
6 custom_table = do.call("rbind", apply(experiments_found, 1, function(experiment){
7   experiment_id = experiment[1]
8   # Obtain the information about the experiment_id
9   info = deepblue_info(experiment_id)
10
11   # Print the experiment name, project, biosource, and epigenetic mark.
12   with(info, { data.frame(name = name, project = project,
13     biosource = sample_info$biosource_name, epigenetic_mark = epigenetic_mark)
14   })
15 })))
16
17 head(custom_table)
18
19 #           name           project biosource epigenetic_mark
20 # 1 E038-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac
21 # 2 E047-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac
22 # 3 E048-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac
23 # 4 E037-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac
24 # 5 E045-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac
25 # 6 E040-H3K27ac.narrowPeak.bed Roadmap Epigenomics BL00D H3K27ac

```

Listing 5.1: Command *search* example, followed by the execution of the command *info*.

The example in Listing 5.2 demonstrates how epigenomic data can be selected with the command *deepblue_select_experiments* and subsequently, how the selected regions can be counted with the command *deepblue_count_regions*. Workflows in *DeepBlueR* follows the structure presented in the *DeepBlue API* Section (4.5), where commands are connected by *query IDs*.

The *deepblue_count_regions* command is executed asynchronously: the user receives a *request ID* and should check the status of this request. In contrast to the *DeepBlue API* command *get_request_data*, *DeepBlueR* contains the package-specific command *deepblue_download_request_data* that waits for the processing to finish and then downloads its data. Moreover, this command converts the downloaded regions to *GenomicRanges* objects.

```

1 query_id = deepblue_select_experiments(
2   experiment_name=c("BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38.20150527.bed",
3     "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"))
4 # Count how many regions where selected
5 request_id = deepblue_count_regions(query_id=query_id)
6 # Download the request data as soon as processing is finished
7 requested_data = deepblue_download_request_data(request_id=request_id)
8 print(paste("The selected experiments have", requested_data, "regions."))

```

Listing 5.2: Example of the *select_experiments* and *count_regions* commands.

Listing 5.3 contains a complete example of selecting experiments and genes, filtering by the regions content and by overlapping, extending, and downloading the regions. It uses the *deepblue_select_experiments* command for selecting genomic regions from two specific experiments that are in chromosome 1, position 0 to 50,000,000. Then, it generates promoters regions, first loading all the genes from *GENCODE v23*, filtering them by

the gene attribute *gene_type* matching the string *protein_coding*, and finally, generating flanking regions that start 2500 *bp* before the *TSS*, and are 2000 *bp* long. The selected experiments' regions are filtered by intersecting them with the promoter regions. Finally, the resulting regions are obtained through the command *deepblue_get_regions*.

```

1 query_id = deepblue_select_experiments(
2   experiment_name = c("BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38.20150527.bed",
3   "S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed"),
4   chromosome="chr1", start=0, end=50000000)
5
6 q_genes = deepblue_select_genes(gene_model="gencode v23")
7
8 q_protein_genes = deepblue_filter_regions(query_id=q_genes,
9   field="@GENE_ATTRIBUTE(gene_type)", operation="==",
10  value="protein_coding", type="string")
11
12 promoters_id = deepblue_flank(query_id=q_protein_genes,
13   start=-2500, length=2000, use_strand=TRUE)
14
15 intersect_id = deepblue_intersection(
16   query_data_id=query_id, query_filter_id=promoters_id)
17
18 request_id = deepblue_get_regions(
19   query_id=intersect_id,
20   output_format="CHROMOSOME,START,END,SIGNAL_VALUE,PEAK,@NAME,@BIOSOURCE")
21
22 regions = deepblue_download_request_data(request_id=request_id)
23 regions
24
25 ## GRanges object with 226 ranges and 4 metadata columns:
26 ##           seqnames          ranges strand | SIGNAL_VALUE      PEAK
27 ##           <Rle>             <IRanges>  <Rle> | <character> <integer>
28 ## [1]      chr1 [ 923976,  924329]      * |      4.7201      109
29 ## [2]      chr1 [1019133, 1019366]      * |      4.4460      156
30 ## ...      ...      ...      ...      ...      ...
31 ## [225]     chr1 [46307396, 46307685]      * |      4.5767      142
32 ## [226]     chr1 [47333183, 47335172]      * |     18.9772      857
33 ##           @NAME      @BIOSOURCE
34 ##           <character> <character>
35 ## [1] S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed myeloid cell
36 ## [2] S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed myeloid cell
37 ## ...      ...      ...
38 ## [225] BL-2_c01.ERX297416.H3K27ac.bwa.GRCh38.20150527.bed BL-2
39 ## [226] S008SGH1.ERX406923.H3K27ac.bwa.GRCh38.20150728.bed myeloid cell

```

Listing 5.3: Example of selecting, filtering, and downloading epigenomic data using the *deepblue_select_experiments*, generating gene promoters dynamically by the commands *deepblue_select_genes* and *deepblue_flank* and filtering regions with *deepblue_filter_regions* and *deepblue_intersection*.

It is straightforward to import the DeepBlue downloaded regions into other *R/Bioconductor* packages because *DeepBlueR* transparently converts them to *GenomicRanges* region sets. Listing 5.4 uses the *GViz* library for displaying the downloaded genomic regions. It combines the regions in two groups based on the regions *@BIOSOURCE*(myeloid or BL-2 cells) content, and display the regions of a segment of the chromosome 1 (Figure 5.1).

```

1 library(Gviz)
2 atrack <- AnnotationTrack(regions, name = "Intersecting regions",
3   group = regions$`@BIOSOURCE`, genome="hg38")
4 gtrack <- GenomeAxisTrack()
5 itrack <- IdeogramTrack(genome = "hg38", chromosome = "chr1")
6 plotTracks(list(itrack, atrack, gtrack), groupAnnotation="group", fontsize=18,
7   background.panel = "#FFFEDB", background.title = "darkblue")

```

Listing 5.4: Connecting downloaded regions to GViz library.

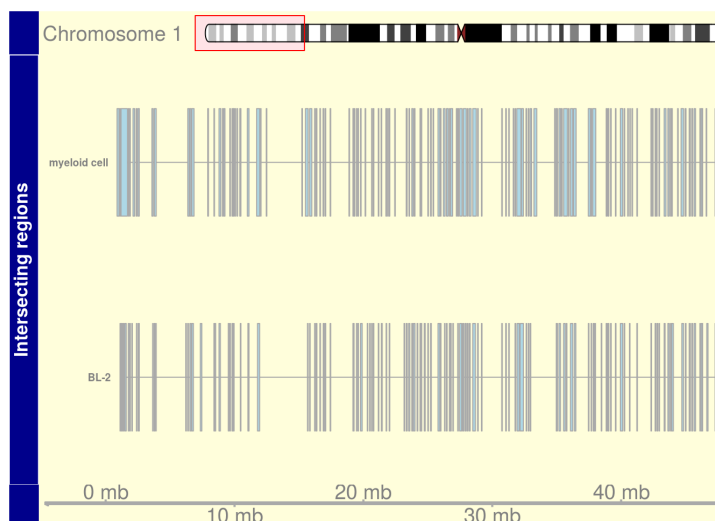


Figure 5.1: Constructed plot containing the retrieved genomic regions divided in two groups by their BioSource: myeloid and BL-2 cells.

Listing 5.5 aggregates genomic regions into tiling regions of 100,000 bps length. It uses the `deepblue_tiling_regions` command to generate the tiling regions and the command `deepblue_aggregate` summarizes the `query_id` regions using their values stored in the column `VALUE` and the CpG island regions as regions summary boundaries. The resulting regions are visualized using the GViz library (Figure 5.2), where it is possible to observe the missing data in the chromosome centromere which is generally difficult to map due to repetitive regions.

```

1 # Select experiment data:
2 query_id = deepblue_select_experiments(
3   experiment=c("GC_T14_10.CPG_methylation_calls.bs_call.GRCh38.20160531.wig"),
4   chromosome="chr1")
5
6 # Tiling regions of 100.000 base pairs
7 tiling_id = deepblue_tiling_regions(size=100000,
8   genome="GRCh38", chromosome="chr1")
9
10 # Aggregate
11 overlapped = deepblue_aggregate (data_id=query_id,
12   ranges_id=tiling_id, column="VALUE")
13
14 # Retrieve the experiments data (The @NAME meta-column is used to include the
15 # experiment name and @BIOSOURCE for experiment's biosource)
16 request_id = deepblue_get_regions(query_id=overlapped,
17   output_format="CHROMOSOME,START,END,@AGG.MEAN,@AGG.SD")
18
19 regions = deepblue_download_request_data(request_id=request_id)
20 regions
21
22 ## GRanges object with 2489 ranges and 2 metadata columns:
23 ##           seqnames           ranges strand | @AGG.MEAN  @AGG.SD
24 ##           <Rle>             <IRanges> <Rle> | <numeric> <numeric>
25 ##      [1]    chr1             0-100000    * |    0.6677    0.3639
26 ##      [2]    chr1          100000-200000    * |    0.8358    0.2414
27 ##      ...      ...              ...      ... |    ...      ...
28 ##    [2488]   chr1 248700000-248800000    * |    0.8425    0.1846
29 ##    [2489]   chr1 248800000-248900000    * |    0.6572    0.4079
30 ##      -----
31
32 library(ggplot2)
33 plot_data <- as.data.frame(regions)
34 plot_data[,grepl("X.", colnames(plot_data))] <-
35   apply(plot_data[,grepl("X.", colnames(plot_data))], 2, as.numeric)
36 AGG.plot <- ggplot(plot_data, aes(start)) +
37   geom_ribbon(aes(ymin = X.AGG.MEAN - (X.AGG.SD / 2),
38     ymax = X.AGG.MEAN + (X.AGG.SD / 2)), fill = "grey70") +
39   geom_line(aes(y = X.AGG.MEAN))
40 print(AGG.plot)

```

Listing 5.5: Command *score_matrix* example.

DeepBlueR also integrates the results of the *deepblue_score_matrix* command into the R/Bioconductor environment. Listing 5.6 presents an example where the regions data of nine experiments are aggregated using the CpG islands annotation as boundaries. The resulting score matrix is plotted using the *ggplot2* library, and the result is displayed in Figure 5.3.

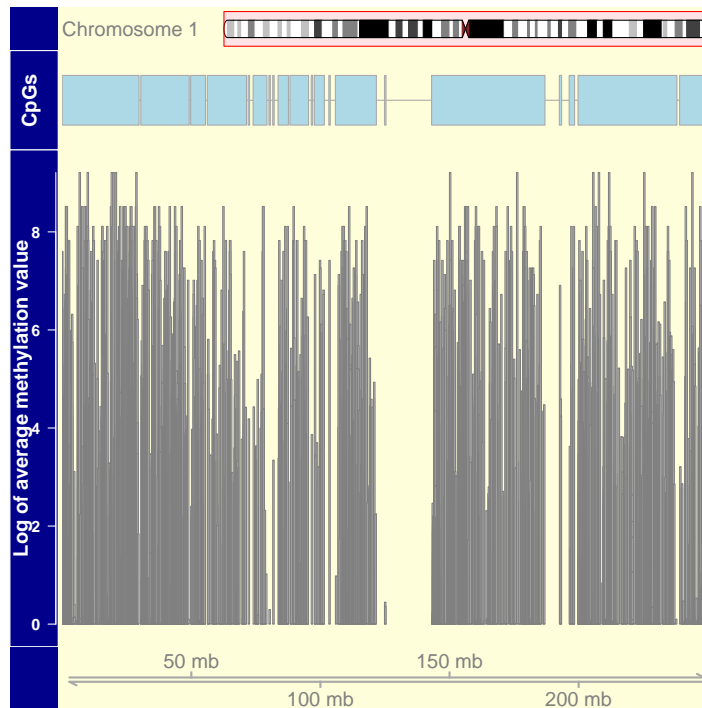


Figure 5.2: Constructed plot containing the resulting regions. The chromosomal location is depicted on the x -axis and the log of the average DNA methylation value is depicted on the y -axis.

```

1 experiments =
2   c("GC_T14_10.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
3     "C003N351.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
4     "C005VG51.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
5     "S002R551.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
6     "NBC_NC11_41.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
7     "bmPCs-V156.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
8     "S00BS451.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
9     "S00D1DA1.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
10    "S00D39A1.CPG_methylation_calls.bs_call.GRCh38.20160531.wig")
11
12 experiments_columns = list()
13 for (experiment_name in experiments) {
14   experiments_columns[[experiment_name]] = "VALUE"
15 }
16
17 cpgs = deepblue_select_annotations(
18   annotation_name="Cpg Islands",
19   chromosome="chr22", start=0, end=18000000, genome="GRCh38")
20
21 request_id = deepblue_score_matrix(
22   experiments_columns=experiments_columns,
23   aggregation_function="mean", aggregation_regions_id=cpgs)
24
25 score_matrix = deepblue_download_request_data(request_id=request_id)
26
27 library(ggplot2)
28 score_matrix_plot = tidyr::gather(score_matrix,
29   "experiment", "methylation", -CHROMOSOME, -START, -END)
30 score_matrix_plot$START <- as.factor(score_matrix_plot$START)
31 ggplot(score_matrix_plot, aes(x=START, y=experiment, fill=methylation)) +
32   geom_tile() +
33   theme(axis.text.x=element_text(angle=-90))

```

Listing 5.6: Example of building a score matrix.

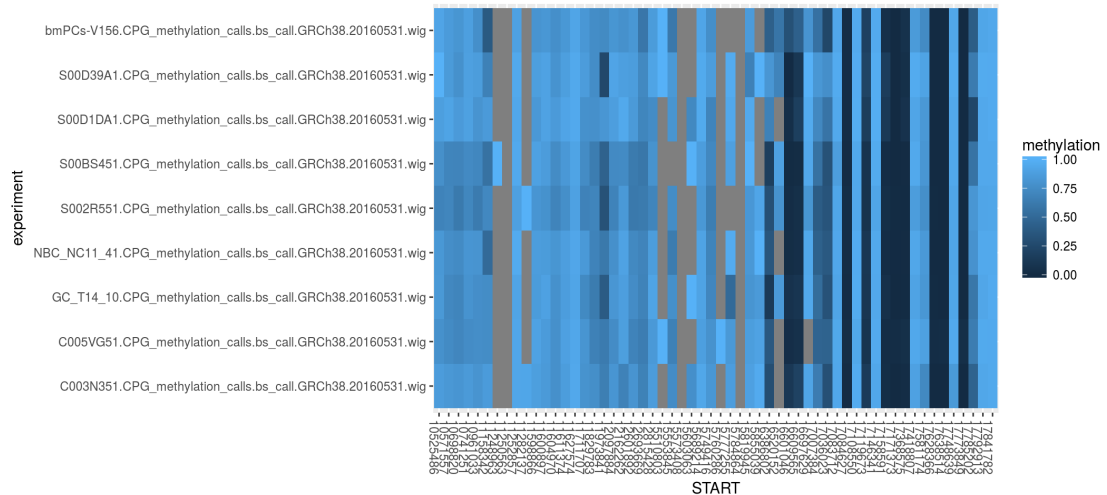


Figure 5.3: Plotting `deepblue_score_matrix` command result with `ggplot`.

5.2.1 Data Export

DeepBlueR allows users to conveniently save results to the local hard disk. All results can be saved as tab-delimited files using the `deepblue_export_tab` command. For example, a user can save the score matrix generated in Listing 5.6 executing the command: `deepblue_export_tab(score_matrix, file.name = "processed_score_matrix")`.

Results obtained with the command `deepblue_get_regions` are of the type *GenomicRanges* and can be exported as tab-delimited files preserving all columns or as BED files, where a specific column can optionally be selected to populate the column *score* presented in the BED file specification. To demonstrate this feature, the result from the example (Listing 5.5) is exported to the local hard disk in Listing 5.7.

```
1 request_id = deepblue_get_regions(query_id=overlapped output_format="CHROMOSOME,START,END,
2   @AGG.MEAN,@AGG.SD")
3 regions = deepblue_download_request_data(request_id=request_id)
4 deepblue_export_bed(regions,
5   file.name = "my_tiling_regions",
6   score.field = "@AGG.MEAN")
```

Listing 5.7: Example of exporting (epi)genomic data to the local hard disk using the command `deepblue_export_bed`.

Furthermore, metadata associated with an entity can be stored locally using the `deepblue_export_meta_data` command. Listing 5.8 demonstrates this feature by first obtaining an experiment *ID* by the command `deepblue_name_to_id` and then exporting its metadata through the command `deepblue_export_meta_data`. The same command can be used to export the metadata of other entities, such as BioSources or processing requests.

```
1 exp_id <- deepblue_name_to_id(
2   "GC_T14_10.CPG_methylation_calls.bs_call.GRCh38.20160531.wig",
3   collection = "experiments")$id
4
5 deepblue_export_meta_data(exp_id, file.name = "GC_T14")
```

Listing 5.8: Example of exporting metadata with the command `deepblue_export_meta_data`.

The `deepblue_export_meta_data` command can export the metadata of a lists of IDs:

```
deepblue_export_meta_data(list("e30035", "e30036"), file.name = "test_export")
```

In some cases, a user performs a sequence of processing requests and wishes to store the processed data locally. For this purpose, the command `deepblue_batch_export` is used to store these results and their associated metadata in the local disk in one straightforward command execution. This command automatically saves each requesting content as soon the *DeepBlue Server* successfully processes it. Listing 5.9 provides an example.

```

1 experiments = deepblue_list_experiments(type="peaks", epigenetic_mark="H3K4me3",
2     biosource=c("inflammatory macrophage", "macrophage"),
3     project="BLUEPRINT Epigenome")
4 experiment_names = deepblue_extract_names(experiments)
5
6 request_ids = foreach(experiment = experiment_names) %do%{
7     query_id = deepblue_select_experiments(experiment_name = experiment,
8         chromosome = "chr21")
9
10    request_id = deepblue_get_regions(query_id=query_id,
11        output_format = "CHROMOSOME,START,END")
12 }
13 request_data = deepblue_batch_export_results(request_ids,
14     target.directory = "BLUEPRINT macrophages chr21")

```

Listing 5.9: Example of storing multiple requests data using the command `deepblue_batch_export`.

5.2.2 Data Caching

DeepBlueR also provides a data caching mechanism in that it automatically stores the data downloaded from the *DeepBlue Server*. The caching has proven useful when a researcher performs a data analysis which requests the same data often from the *DeepBlue Server* or in a setting of limited network bandwidth. It is possible to switch off caching, configure the cache size, and delete an individual *DeepBlue Request* data or clear the entire cache.

5.3 Use cases

This section presents three use cases that illustrate the functionalities of *DeepBlueR*. The use cases are: (i) clustering blood samples by their DNA methylation data; (ii) predicting gene expression based on histone marks; (iii) constructing cell type signatures; (iv) metadata and batch effect analysis.

5.3.1 Clustering blood samples by their DNA methylation data

DNA methylation is a well suited epigenetic mark for studying cellular differentiation because its patterns are cell type-specific and retain an epigenetic memory of a cell's developmental history (Farlik *et al.* 2016). Using machine learning methods, Farlik *et al.* 2016 performed an in vivo dissection of human hematopoiesis, reconstructing the human blood cells genealogy, from *Hematopoietic Stem Cell* (HSC) to Megakaryocytes, Monocytes, Neutrophils, B Cells, and T Cells.

This use case demonstrates how *DeepBlueR* can be used to gather the necessary data to perform a simpler analysis in just a few lines of code. It generates an overview heatmap of summarized genomic regions data in more than 200 BLUEPRINT DNA methylation experiments and clusters the samples by their DNA methylation profile. It is divided into three parts: (i) the DNA methylation experiments (*deepblue_list_experiments*) are selected; (ii) a score matrix (*deepblue_score_matrix*) is constructed using the average DNA methylation score (beta value) in all regulatory elements defined in the *BLUEPRINT regulatory build* (modified from [Zerbino et al. 2015](#)) across all BLUEPRINT samples. Each experiment data file has between 120 to 180 megabytes, totaling more than 20 gigabytes of data that are processed directly on the server; (iii) the package *ggplot2* is used to create a heatmap using the score matrix data processed by the *DeepBlue Server*, showing the most variable regions (rows) across samples (columns). Moreover, it clusters samples by their pairwise Spearman correlation coefficients.

Figure 5.4 displays the clustering result. Through visual inspection, it is possible to see that similar biosources are clustered together, and that clusters reflect the biological meaning defined in the haematopoiesis process. The complete source code of this use case is available in Section A.3.2.

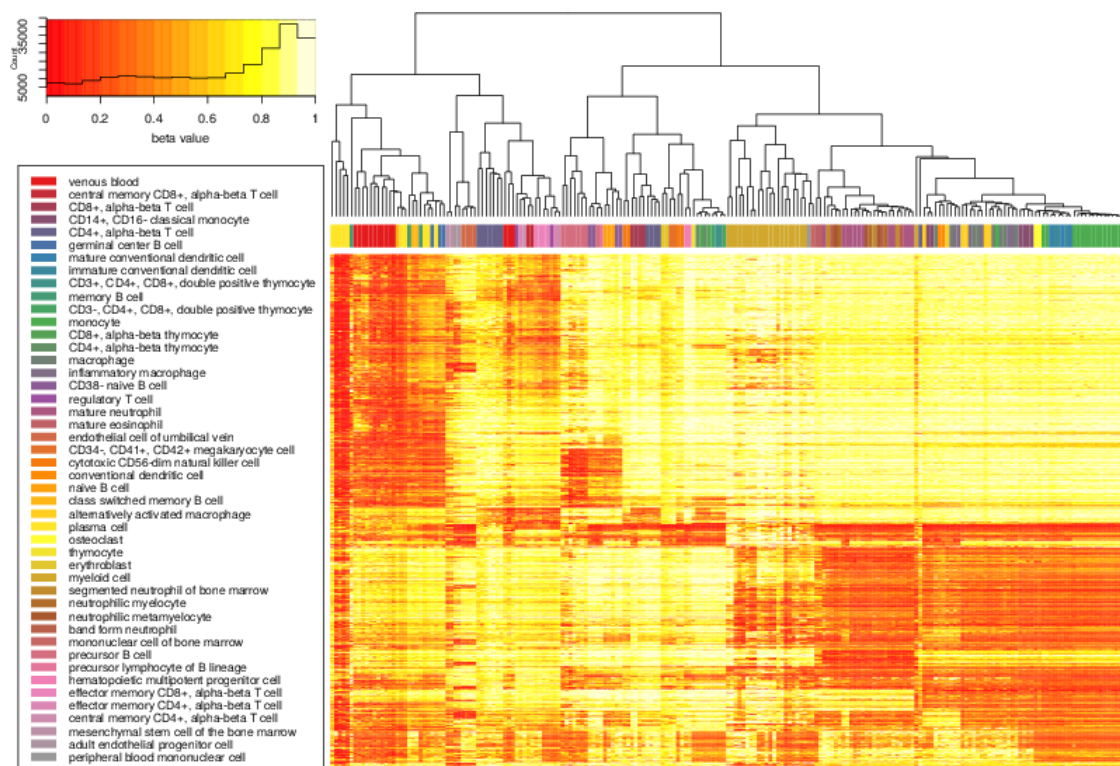


Figure 5.4: Clustering blood samples by their DNA methylation data: This heatmap contains the average DNA methylation values of variable (variance > 0.05) regions of the BLUEPRINT regulatory build across 206 BLUEPRINT epigenomes comprising 47 different cell types.

This use case demonstrates how DNA methylation data can be gathered effortlessly in *R/Bioconductor* via the *DeepBlueR* package. Moreover, it shows how visualization techniques in *R* can be leveraged to study this data. However, this is just a stepping stone

for further analysis using *R*'s rich ecosystem for statistical learning as it is seen in the following use case.

5.3.2 Predicting gene expression based on histone marks

Predicting gene expression from epigenomic data sources is a biological problem that helps researchers to obtain a better understanding of how epigenetic mechanisms contribute to regulating gene expression. For instance, it is well known that some histone marks act as repressors while others enhance gene expression (Table 2.1). This use case demonstrates how to use *DeepBlueR* to reaffirm this knowledge.

This use case uses histone marks data for predicting gene expression. Moreover, it identifies the most informative histone marks and their genomic location, i.e. the regulatory regions encompassing the gene that impact at most in this gene expression. Gene expression prediction based on histone marks is a well-established problem (Karlić *et al.* 2010). More recently, a new approach using deep learning (Singh *et al.* 2016), achieved an *AUC* of 0.80 for classifying the gene expression level as high and low. This use case also predict the gene expressions from histone marks, but with a simpler statistical learning method, the elastic net (J. Friedman *et al.* 2009), implemented in *R* in the *glmnet* (J. H. Friedman *et al.* n.d.) package.

This use case uses six different epigenetic marks: *H3K4me3*, *H3K36me3*, *H3K27ac*, *H3K4me1*, *H3K27me3*, *H3K9me3* from six samples from the BioSource *CD4-positive, alpha-beta T cell*, totaling 36 *ChIP-seq* signal data files. Gene expression data was obtained from the same samples, in total, six files containing the gene expression in *TPM* values. Finally, this use case uses *GENCODE v22* for gene annotations. The source code of this use case is presented in Appendix A.3.2 and it is divided into the following parts:

- Obtain promoter region bins:
 - Select promoter regions (2500 *bp* before the *TSS*, 5000 *bp* long)
 - Split genome into 100*bp* bins
 - Intersect promoter regions with genome bins
- Mean aggregation of the *ChIP-seq* signal data in each bin and request result
- Request *TPM* values of all protein coding genes
- Download gene expression and histone data
- Match gene expression and *ChIP-seq* data
- For each histone mark:
 - Use the histone and gene expression data as input of the *elasticnet* classifier with the following parameters:
 - * $\alpha = 0.5$
 - * 10 times cross validation
 - * function *cv.glmnet* of the *glmnet* package
 - Obtain the prediction result and *AUC* values
 - Discretize the gene expression predicted values in high and low expression

Figure 5.5 shows the most informative histone marks with their respective *AUC*. The *AUC* values are between 0.772 to 0.873. Figure 5.6 shows the most informative bins for predicting gene expression. It is important to note that the bins coefficient correlate to the known biological histone mark code, where the bins of repressive histone

marks ($H3K9me3$ and $H3K27me3$) have a negative coefficient, and active histone marks ($H3K4me3$ and $H3K4me1$) have mainly positive coefficients.

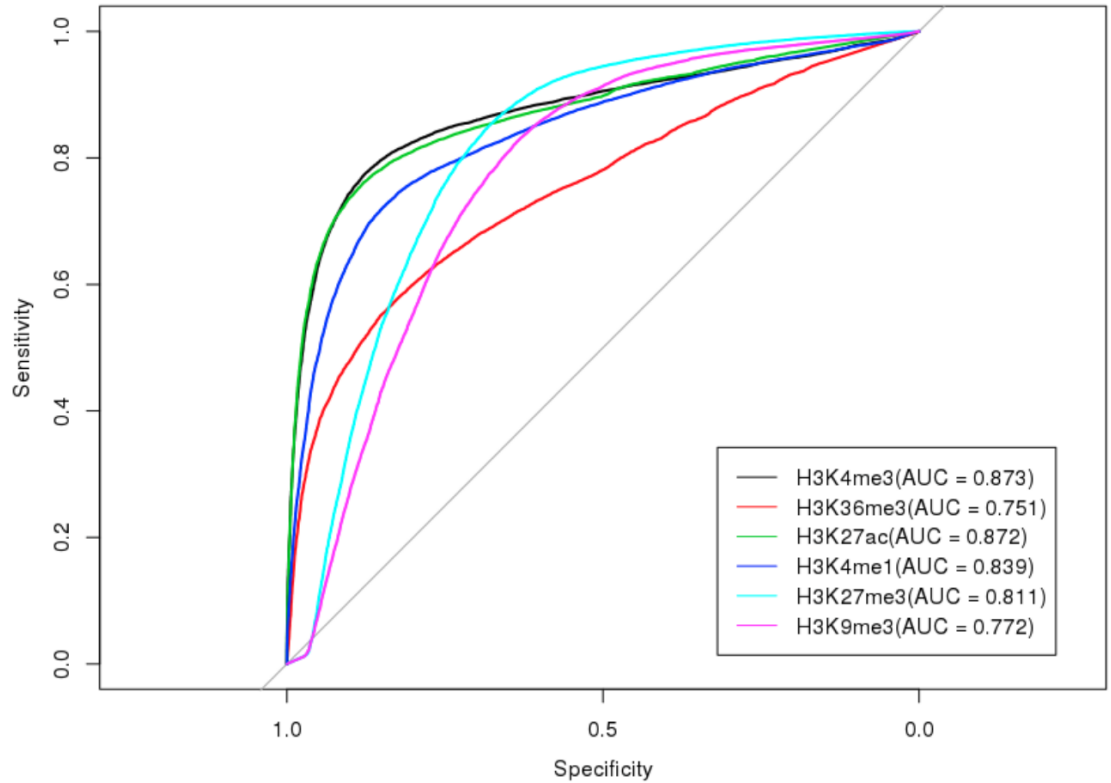


Figure 5.5: Histone marks AUC for gene expression prediction.

This use case provides a better performance than the deep learning approach reported in Singh *et al.* 2016 with AUC of 0.80). One reason may lie in the high data quality used. But another reason for the good performance could be that all of the samples were from the same or a similar cell type. With a result of such a high quality, this use case is a starting point for an in-depth comparison of different machine learning strategies for predicting gene expression. Moreover, this analysis is limited to only six samples, having more data from different BioSources may yield a better understanding of cell type specific effects of histone marks on gene regulation.

5.3.3 Constructing cell signatures

Establishing a comprehensive list of genomic regions and marks can help to identify and lead to the development of cell type-specific epigenomic biomarkers (García-Giménez *et al.* 2016; Toska and Sanz 2016). As an example, DNA methylation of different regions of the genome is used as a biomarker in prostate cancer (Ferro *et al.* 2017; Ramalho-Carvalho *et al.* 2016), lung cancer (Sandoval and Serra 2016; Diaz-Lagares *et al.* 2016), breast cancer (Cervera *et al.* 2016; Tang *et al.* 2016), asthma and allergies (DeVries and Vercelli 2016), and obesity (Crujeiras and Diaz-Lagares 2016). Other epigenetic marks are also suitable as biomarkers, like histone modifications, and Micro RNA (*miRNA*) (García-Giménez 2015).

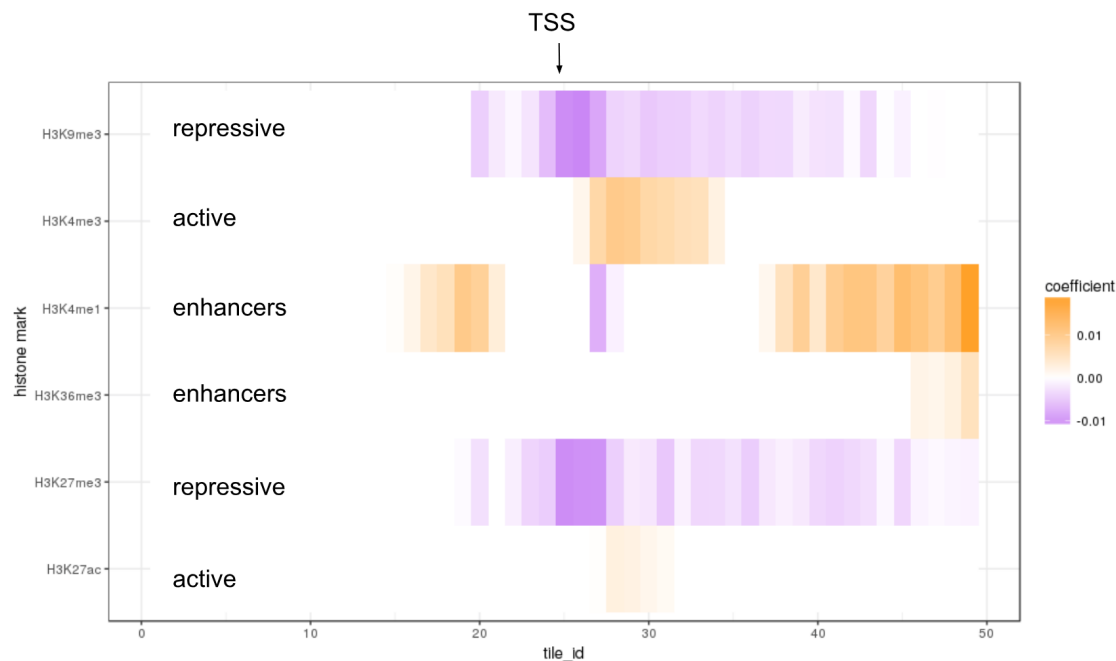


Figure 5.6: Most informative genome location for predicting gene expression. The bins coefficient correlate to the known biological histone mark code.

The current development of biomarkers requires manual work and a priori knowledge of the potential genomic locations and epigenetic marks. For improving the development of new biomarkers, their development process must be systematized and automated. For this purpose, the first step is to generate signatures of the cell types, based on cell-type specific regions.

This use case demonstrates the process of systematically generating cell type signatures. It uses DNA methylation data from the BLUEPRINT project for generation of cell type signatures of blood cell types. For this purpose, 167 experiment files were used. Listing A.10 contains the source code of this use case. Due to its complexity, this use case is divided into three main tasks:

1. Select cell types
2. Obtain region-specific DNA methylation data from selected experiments
3. Systematic selection of cell type specific biomarkers

The first task (Listing A.10-lines 1 to 59) loads experiments and their metadata, all in the form of BLUEPRINT DNA methylation files, and writes the result in an Microsoft Excel spreadsheet. Next, it pauses the execution, allowing the user to edit the spreadsheet, selecting the experiment files which are necessary. Figure 5.7 shows the number of experiments for each selected cell type.

The second task (Listing A.11) obtains the DNA methylation data from the previously selected experiments. The data is retrieved using the *deepblue_score_matrix* command³. The second task performs the following steps:

1. Summarize the experiments data in a score matrix using the ENSEMBL regulatory build:

³ As a performance statement, it does took approximately two hours for processing all 167 samples.

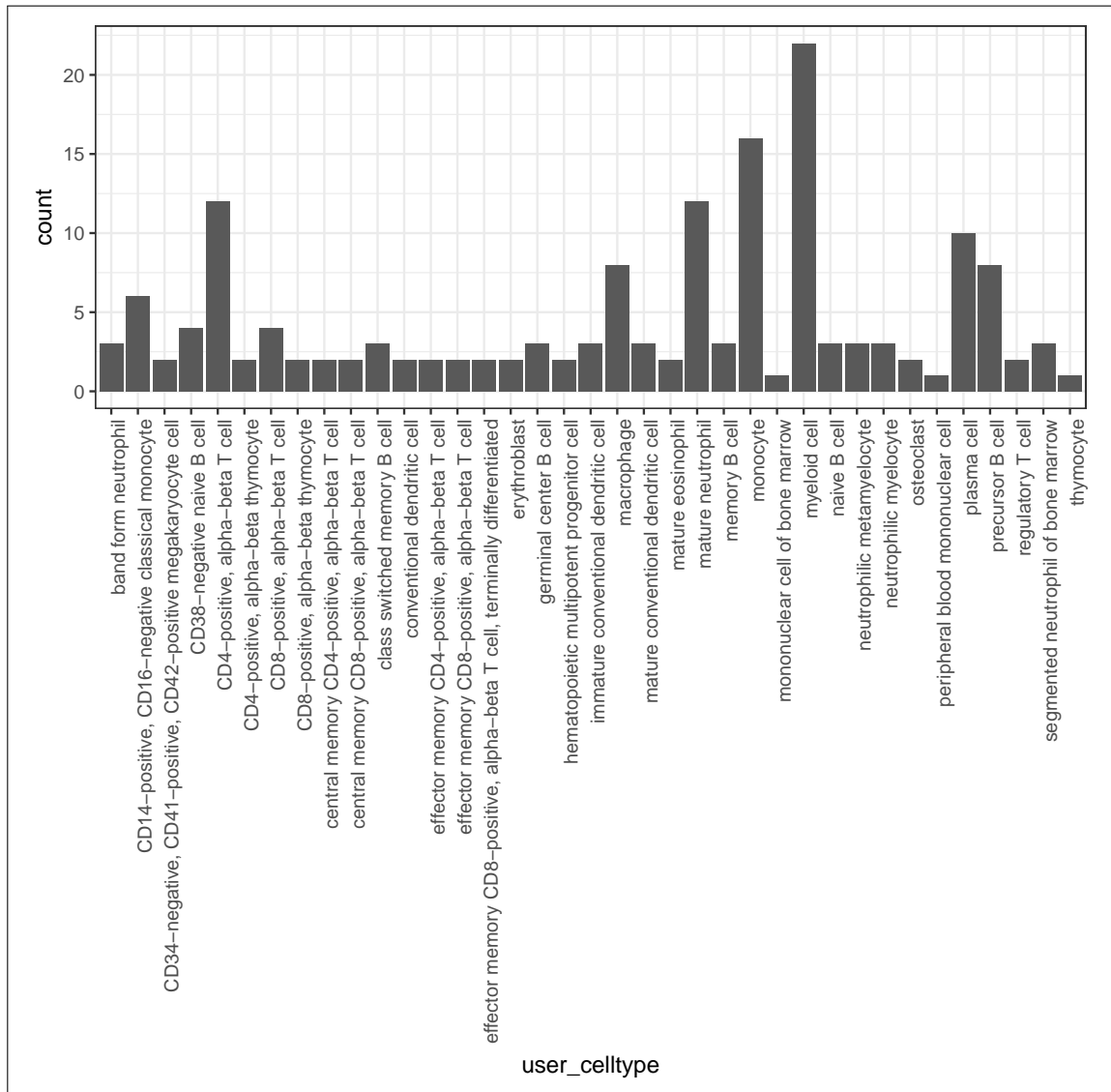


Figure 5.7: Cell types used for generating cell type signatures.

- For each ENSEMBL regulatory region compute the mean and standard deviation of the DNA methylation values for all samples of each cell type
2. Export the resulting score matrix

The third task (Listing A.12) selects the cell type specific biomarkers, i.e., the regions that contain significant differences between the cell types. This process has the following steps:

1. Based on the score matrix produced in the second task, add for each cell type the following calculated measures of cell type specificity:
 - number of cell types in which the region's mean DNA methylation is lower than the region's DNA methylation in the selected cell type
 - number of cell types in which the region's mean DNA methylation minus 1x the standard deviation is lower than the region's mean DNA methylation in the selected cell type

- number of cell types in which the region's mean DNA methylation minus 2x the standard deviation is lower than the region's mean DNA methylation in the selected cell type
2. Rank the regions based on the worst rank of the three metrics (each one ranked individually and then using the row-wise maximum)
 3. Return the top 500 regions that are hypomethylated in the selected cell type based on the consensus ranking
 4. Repeat steps 1 to 3 with a focus on hypermethylated regions in selected cell types

For ensuring the script code correctness, this use case uses automatic testing (Listing A.14) that verifies if the hypo and hyper methylation values are correctly computed.

Figure 5.8 shows the clustering of the experiments using the identified hypomethylated regions, and Figure 5.9 shows the clustering using the hypermethylated regions. In both cases, it is possible to observe a separation between different cell types, showing that those regions are valid signatures.

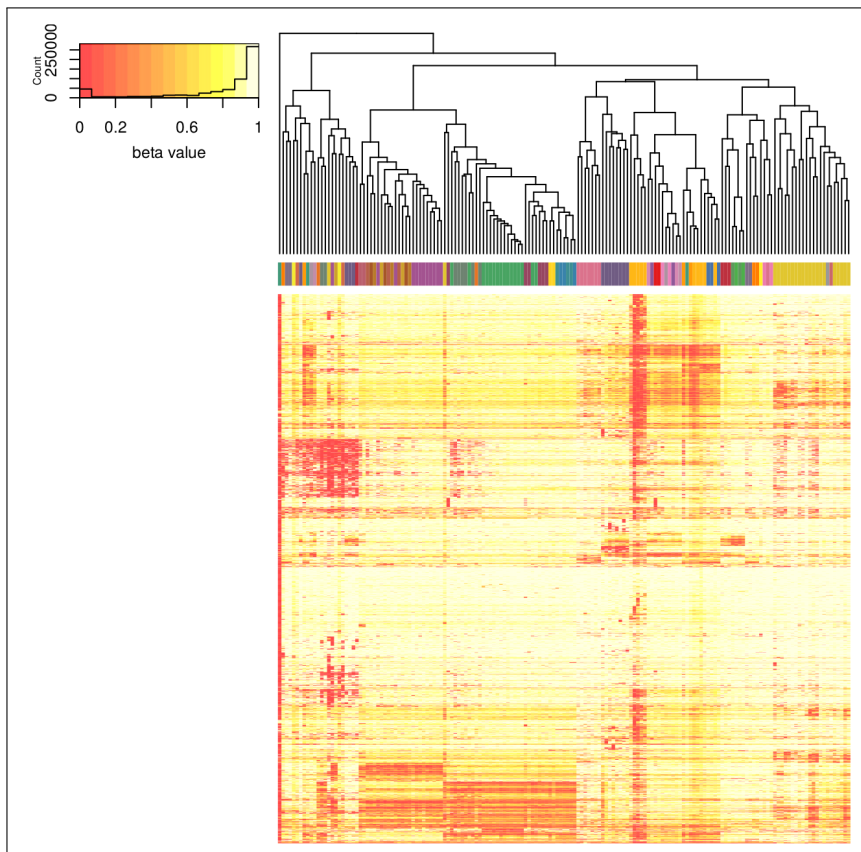


Figure 5.8: Cell types signatures: DNA hypomethylated regions heatmap.

For future improvements, we suggest that modifications such as a pre-filtering for regions with high variance across cell types may improve the results. This use case provides a framework for epigenetic cell type signatures that can be used as initial information for biomarker development.

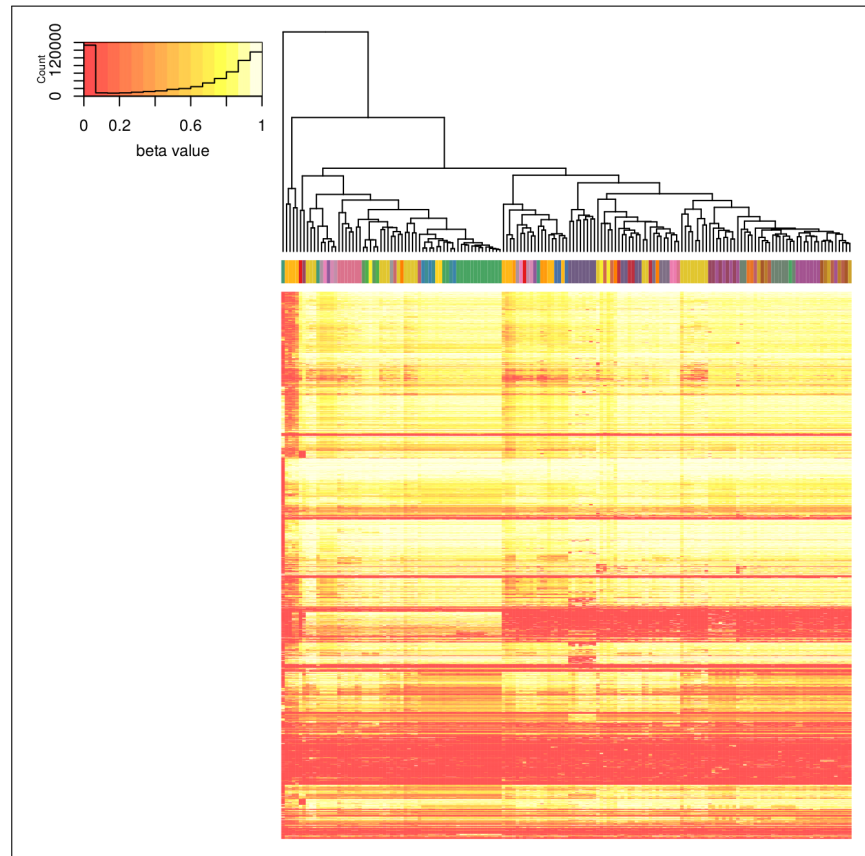


Figure 5.9: Cell types signatures: DNA hypermethylated regions heatmaps chart.

5.3.4 Metadata and batch effect analysis

Metadata are an Achilles' heel of reproducible research: resources are spent generating data, and this data can become useless if they are not correctly annotated with complete and standardized metadata. Even complete metadata may have problems, for example, inconsistent use of terms (e.g., *ChIPseq* vs. *ChIP-seq*), files formatting or encoding problems, and different vocabularies for annotating the metadata fields. As a result, data analysis often focuses on a small set of well-described datasets.

The problem of sharing and maintaining consistent metadata is not new. There are many software packages developed to tackle exactly these tasks, usually called *Laboratory Information Management System (LIMS)*. Such software is available commercially or open-source. For example OpenLabFramework⁴ is an open-source *LIMS* for managing samples, storing information about media and buffers, and using smart QR bar-codes. *LIMS* systems work well if the different laboratories involved in the research agree upon their usage in the beginning of the project. However, usually, each laboratory starts on using its own system for storing information, and consequently, each laboratory member uses different keywords for describing the same process. Later on, when the datasets are merged in one repository, organizing and having consensus between all different metadata is a challenging task, especially if controlled keys and controlled vocabularies were not enforced from the beginning of the project. Farnel and Shiri 2014 describe that

⁴ <http://nanocan.github.io/OpenLabFramework/>

the use of controlled vocabularies and unique identifiers are central components for data and metadata in scientific research, and having a controlled vocabulary for each factor is extremely helpful when applying the mathematical models of batch effect detection and analysis on the data, as consistency in metadata is required for these models to work.

The importance of correct metadata comes with the fact that research has been moving towards larger multi-institutional collaborations, which are becoming essential for producing large and robust datasets that are crucial in answering important scientific questions. However, collaborative studies add more difficulties in managing, sharing and keeping data consistent between different laboratories and teams, and reusing the generated data is profoundly affected and influenced by the level of metadata consistency and interoperability (Garoufallou and Papatheodorou 2014). Again, keeping metadata in a structured and consistent form is a challenging task, especially in large consortia. Typically, these issues are solved manually and only partially.

In this work, the *DEEP* project is used as an example for of metadata cleanup. The project has produced high-throughput data that involves different laboratories, instruments, and software, which can introduce variation and batch effects in the data (Leek *et al.* 2010). The *DeepBlueR* package is used for querying the metadata of the *DEEP* experiment files.

The initial goal of this work was to analyze data generation batch effects systematically, but it was hindered by usual problems with the metadata content that must be dealt with before further studies. Moreover, it faced the following issues:

- Wrong Formatting Some metadata files differ from the previously agreed format (two-columns, tab-separated files, or comma separated files (CSV)). The format consistency is necessary to be able to read, merge, and analyze these data in automatic pipelines. Content might be misinterpreted, causing execution problems in the pipeline, such as loss of time or data due to being unable read the metadata content.
- Different Encoding As many different teams and laboratories are involved in the research project, different operating systems have been used to generate the metadata files. Different operating systems might have different text files encodings, for example, *UTF-8*, *UTF-16*, or *ISO-8859*. Again, this might cause problems when trying to read the content of different metadata files automatically.
- Missing Values Human or machine errors are inevitable: humans may forget to document an experiment process step, or mix the values of a specific buffer concentration can happen. Besides, technical difficulties, like hardware failures, might cause one or more observation to go missing or introduce errors.
- Inconsistent nomenclatures Inconsistency between the metadata terms names may happen when no controlled vocabulary scheme is enforced during the initial project phase. For example, *ChIPseq* vs *ChIP-seq* and *DNase* vs *DNase*. For humans, this is not a problem, but problems occur when using the metadata in automated pipelines. These discrepant values might be considered as two different factors rather than one, generating inconsistencies in the data processing and analysis workflows.

All these issues directly affect the building of statistical models for the detection and correlation of batch effects. For this reason, it proved necessary to clean the metadata before analysis. For fixing the previously listed issues, a semi-automated pipeline was implemented with steps for processing, clearing, and producing a uniform metadata content. As a complement to the metadata clearing pipeline, a tool for analyzing the batch effect from different datasets was developed.

In research that is dealing with massive datasets and many institutions and scientists, considering batch effects, confounding factors, and biases are significant. Batch effects are caused by variability in the data that are introduced by external sources related to the sample preparation, instruments, or data processing. For example gene expression measurement are sensitive to external variables, such as buffers used, laboratory's temperature, or the way of handling the instrument. The same applies to other external causes, such as the conditions for sample storing and preparation, the status of the devices, and software version and parameters used in the data processing. Confounding factors can impact actual correlations and associations in the data, impact which would cause false associations (Skelly *et al.* 2012). Biases can occur during different steps of research, whether it was during planning, measuring and data collection, or statistical models used for downstream data analysis.

These introduced factors can affect data variability and might give false correlations and conclusions when using statistical models during analysis. Batch effects and confounding factors can complicate the process of establishing links and causal effects in the data. However, these factors can be detected, and the data can be adjusted accordingly using statistical models (Leek *et al.* 2010). In this use case, the primary importance of correcting the metadata here is the use for adjusting the data for batch effects and confounding factors.

Metadata clean-up pipeline

The *DEEP* experiments metadata are tab-separated text files with two columns in a key-value fashion. The keys are agreed upon, which helps to merge different metadata files and to compare different experiments and samples. Even in such a straightforward environment, mistakes were made when filling the metadata content and must be fixed before further data analysis.

For solving the previously mentioned metadata issues, a mechanism is required that ensures that the same mistake does not occur again by verifying new metadata content. For addressing the formatting and encoding, a script can fix the problematic files. It is important to note that the developed script is dependent on the data content and can vary from one project to another.

Human interaction is required for fixing missing values. It is crucial to contact the team members responsible for the metadata generation and, with their help, to fill the missing values of the essential metadata factors. However, for inconsistent nomenclatures, a simple method can be used to keep track of the manual changes made to the data, saving these changes and rules where they were applied, for future repetition to the different metadata files that might contain the same mistakes. In addition, having the set of modifications saved separately can help track down the changes made when it is required to revise or modify them.

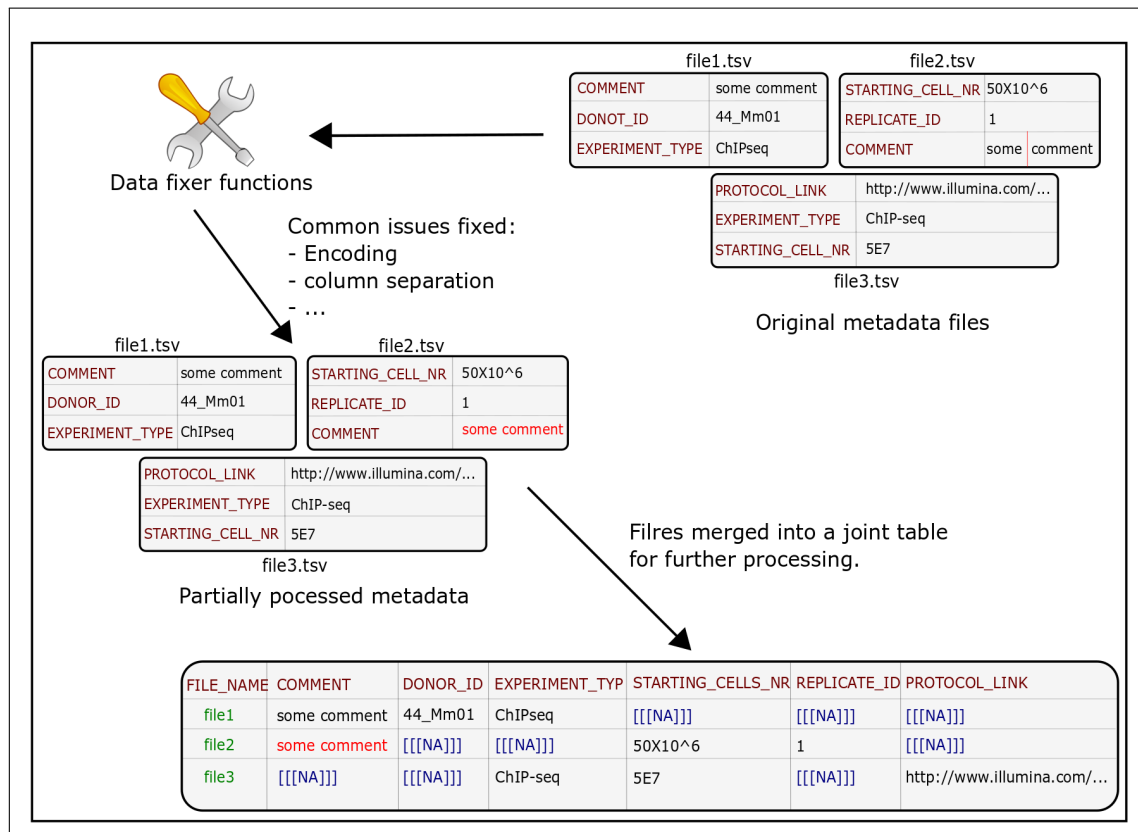


Figure 5.10: Fixing metadata encoding and formating issues: The first step is the fixing of the formatting and encoding problems, e.g., the comment in *file2.tsv* has a tab separator in the value column that was introduced by mistake. Formatting issues are solved by reading the metadata files in the right encoding, and also verifying if characters can be read correctly. The data fixer functions output individual corrected metadata files that are merged into a unique table, which is the input for the further processing made with the *OpenRefine* software.

Figure 5.10 shows the first step of the metadata clearing pipeline: fixing the metadata encoding and format. After, using the data fixer function, a new set of metadata files, with their encoding and content corrected, is generated. The metadata files are then merged into a table that is the input for further processing.

The developed semi-automatic pipeline uses the user-friendly *OpenRefine*⁵ tool for organizing the metadata and documenting the performed changes in a JSON file, which its interface is a web application developed for filtering and manipulating data sheets. This tool was formerly developed by Google to organize disordered data, and to help cleaning and to modify large files content. Using *OpenRefine* brings the following benefits: (i) a visual interactive environment that a non-programmer can easily operate; (ii) tracking of all executed operations. The tool stores the performed operations in a JSON file that can be separately stored, manually edited, and easily re-applied on the data; (iii) the pipeline script easily integrates the generated JSON file that contains the

⁵ <http://openrefine.org/>

operations performed on the metadata. This allows the *OpenRefine* execution externally, automatically performing the operations and correcting metadata files.

Figure 5.11 shows the combined metadata files table that is generated by the previous pipeline step. *OpenRefine* imports this table for further metadata corrections, such as correcting the value *ChIPseq* to the corrected *ChIP-seq* and modifying the value *5E7* to a consistent value of 50×10^6 . The central concept in the *OpenRefine* step is to perform the manual work only once. Later, new metadata files can be cleaned automatically using the previously performed operations that are stored in a JSON file. Besides, this file provides a catalog of manual processes, helping the reproducibility by documenting of what has been changed, and also, easily allowing to include or remove data manipulation operations.

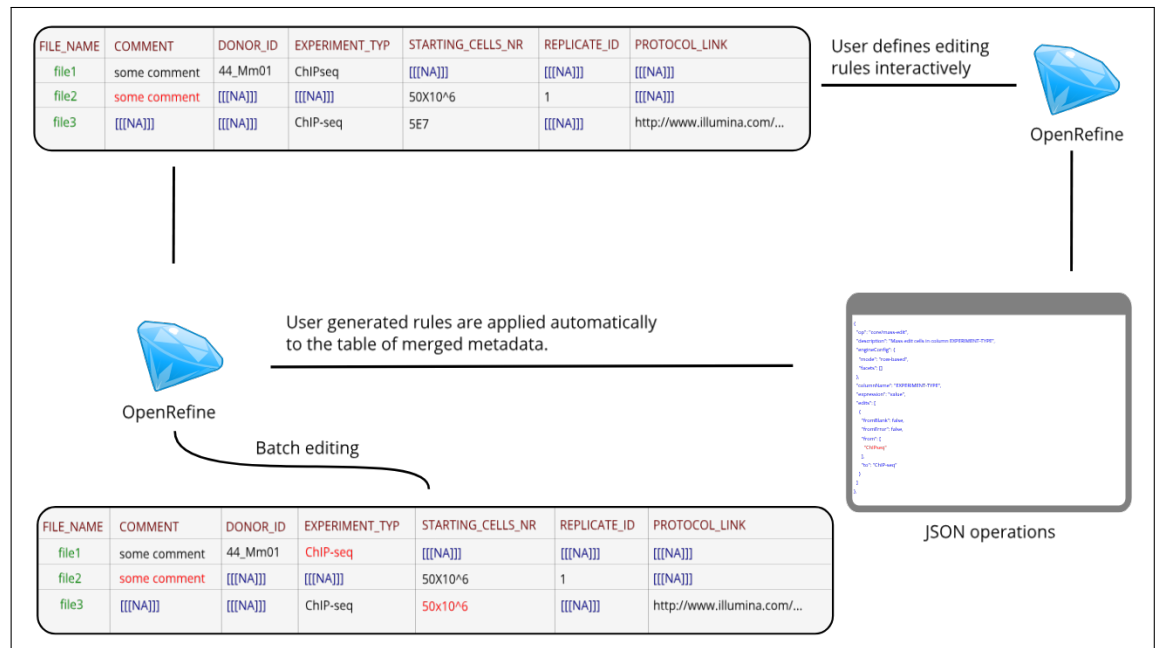


Figure 5.11: Fixing metadata content with *OpenRefine*: users manipulate the metadata content where all steps are stored in a JSON file, which is used in future metadata. The output is a table with fixed metadata values, for example: in the *file1* the value *ChIPseq* was corrected to *ChIP-seq*, and in *file3* the value *5E7* was modified to be consistent with other numerical values as 50×10^6 .

After fixing the files' metadata content with *OpenRefine*, the pipeline converts the table back to individual files. Figure 5.12 shows the pipeline step that automatically evaluates the fixed metadata files, ensuring that their content is correct before they are included in the database. This pipeline step used the fixed metadata files, and two additional files that are manually produced. One file is a regular expressions dictionary, having keys similar to the metadata keys and values that are regular expressions, defining which content is valid for each specific key. The other file contains a black-list of metadata key names which their values content are not regulated.

As previously observed, the use of controlled vocabulary and unique identifiers are central components for ensuring the metadata quality. For this reason, the pipeline uses controlled vocabularies built using the previously fixed metadata files. These controlled vocabularies contain all keys with their content that are not blacklisted or defined by

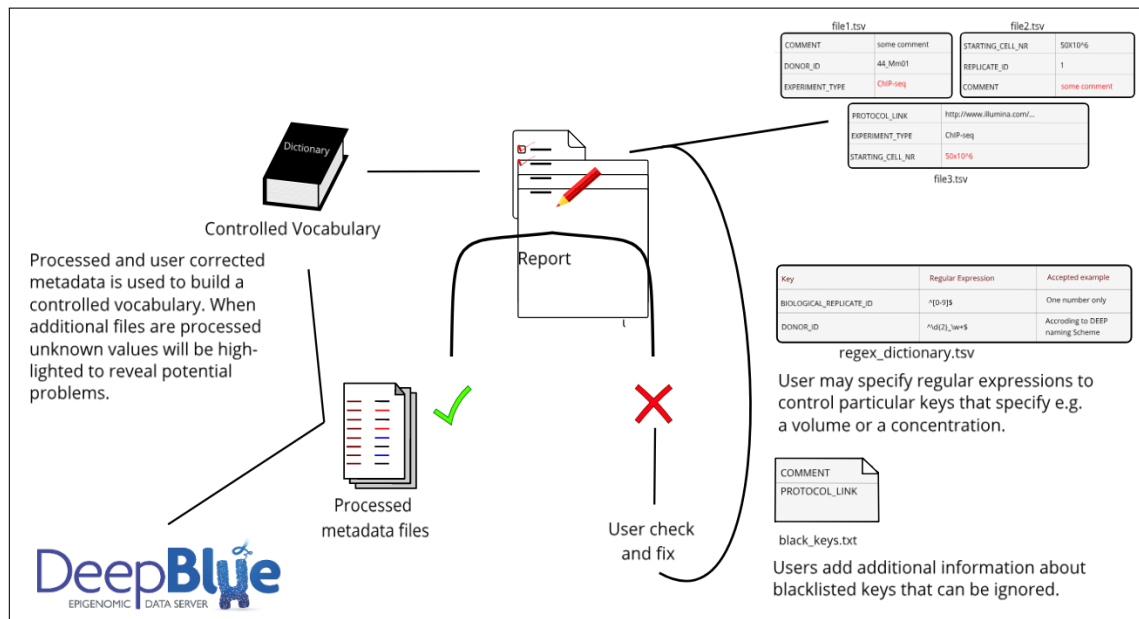


Figure 5.12: Ensuring the correctness of the metadata files: the pipeline converts the table back to individual files and then verifies their content using regular expressions. For example, the key *DONOR_ID* only accepts values that start by two digits followed by an underscore then followed by two or more letters or numbers. If a problem is detected, a user must perform a manual correction. Otherwise, the corrected metadata file is stored into the *DeepBlue Server*.

regular expression. These controlled vocabularies are used for generating automatic reports on the corrected metadata files, which is an essential tool for examining the quality of the metadata files before being stored in a database or shared with other teams. The assumption here is that most new metadata files do not introduce new keys and values but reuse existing ones. Consequently, differences observed here are likely errors.

After the corrected metadata files have been reviewed and approved, they are inserted, with their respective samples and experiments, into the *DeepBlue Server*. As *DeepBlue* enables users to access and operate on relevant epigenomic experimental data, it is crucial that these data contains clean and correct metadata.

Batch effect analysis

As a continuation to the metadata clearing pipeline, a tool for analyzing batch effects in epigenomic data was developed. This tool provides a visual interface where users select data available in the *DeepBlue Server* and easily use one of the three batch effect normalization and analysis packages: *SVA* (Leek and Storey 2007; Leek *et al.* 2012a; Leek *et al.* 2012b), *RUV* (Risso *et al.* 2014), and *ComBat* (W. E. Johnson *et al.* 2007).

The *SVA* package contains methods for removing artifacts both by: (i) identifying and estimating surrogate variables for unknown sources of variation in high-throughput experiments and (ii) directly removing known batch effects using *ComBat*. *RUV* remove unwanted variation that uses factor analysis to adjust for nuisance technical effects, based on counts (or residuals counts) for either negative control genes or negative

	id	name
	All	All
1	e106058	51_Hf03_BICM_Ct_WGBS_S_1.MCSv3.20150424.GRCh37.cpg.filtered.CG.bedgraph
2	e106129	41_Hf14_LiHe_St_WGBS_S_1.MCSv3.20151127.GRCh37.cpg.filtered.CG.bedgraph
3	e105983	41_Hm08_LiHe_St_WGBS_S_1.MCSv3.20151210.GRCh37.cpg.filtered.CG.bedgraph
4	e105820	41_Hf05_LiHe_St_WGBS_S_1.MCSv3.20150424.GRCh37.cpg.filtered.CG.bedgraph

Figure 5.13: Selecting experiments in the batch analysis tool: users can choose a genome and according to the choice, projects are presented. For the *DEEP* data, user can remove the coverage files, or use regular expression for refining the file search.

control samples that is, genes or samples that are not expected to be influenced by the biological covariates of interest (Risso *et al.* 2014).

The *DeepBlueR* package is used to obtain the experiments data and metadata, and to process their data in a score matrix format that is used for the batch analysis cleanup and processing.

The batch analysis tool has a straightforward and intuitive interface. Figure 5.13 shows the interface for selecting the data, and Figure 5.14 shows the information about the metadata content of the selected files.

After selecting the experiments, users must define the selected score matrix parameters, which is computed using the operation *deepblue_score_matrix* (Figure 5.15). The options for processing a score matrix are:

- Regions boundaries:
 - Tiling Regions
 - Genomic Annotations (CpG Island, Promoters)
- Chromosomes:
 - Whole genome
 - Specific chromosomes
- Aggregation function:
 - minimum value
 - maximum value
 - values sum
 - values mean
 - values variance
 - values standard deviance

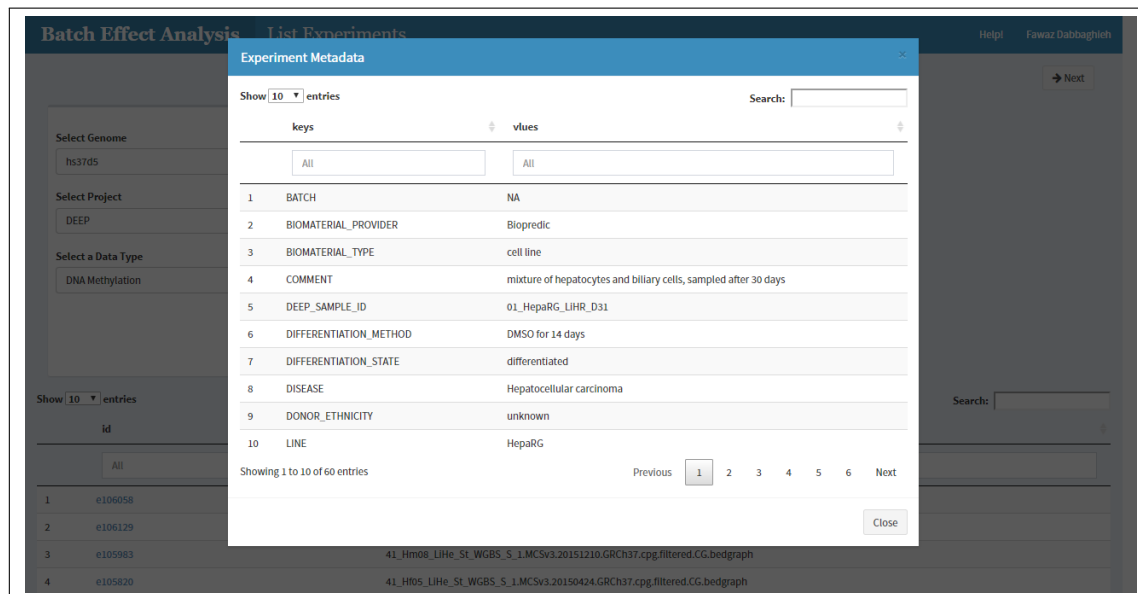


Figure 5.14: Users can select experiments and obtain their metadata content.

- values median
- values count
- boolean - if a value is in such boundary
- Filtering method:
 - Remove incomplete rows
 - Remove rows with low variance (it is possible to define the minimum variance value)

Figure 5.16 shows the interactive *PCA* plot that is available after the score matrix has been processed. The chart allows coloring points based on any metadata, and it can be downloaded as a static *PDF* file or as an interactive web page file.

Figure 5.17 shows the next step in the tool, the batch correction processing. For this purpose, several batch effect correction methods are available: ComBat, SVA, Supervised SVA and RUV. Users can perform several tests and store the results in the *Batch Corrected Matrices* list, and continue the analysis with the selected method. The interface shows the experiments with their corrected values, which can also be downloaded. Optionally, a user can remove outlier experiments before calculating the batch effect.

Figure 5.18 shows the final step, the comparison of the batch effect correction result with the original values. This interface allows the comparison of the results in an interactive chart, containing the data from the original and the corrected matrix next to each other. This chart can be downloaded as a static and interactive file.

Summary

The presented pipeline gives a strategy on how to process inconsistent metadata that many teams and laboratories produce in the context of multi-laboratory and collaborative projects. At the same time, the use of OpenRefine helps to keep track of any changes

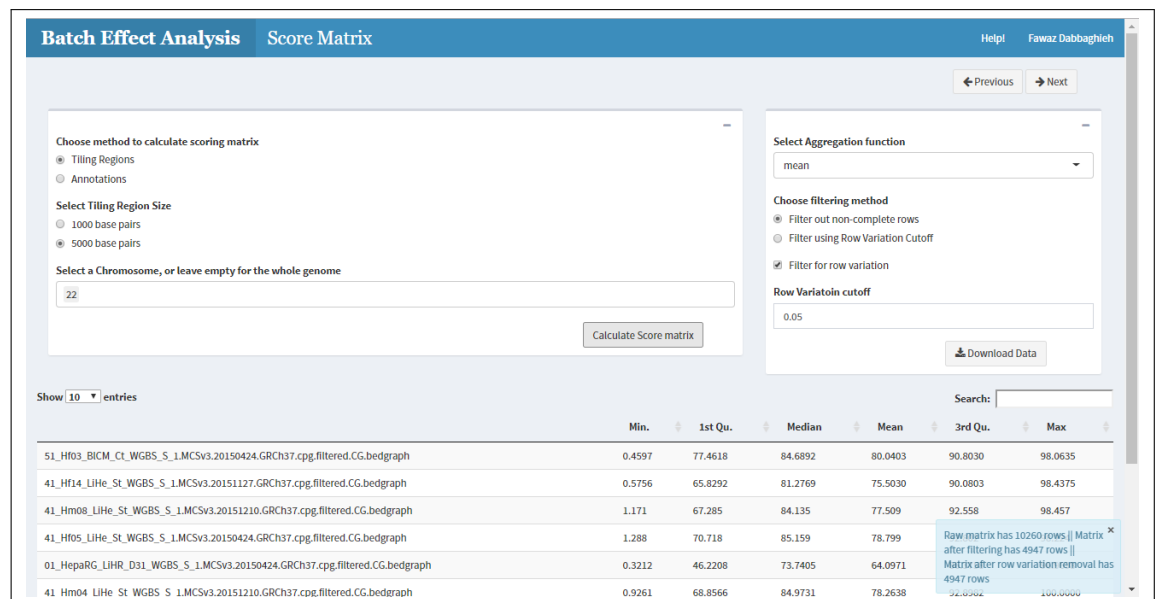


Figure 5.15: Defining the parameters for building the score matrix in the batch analysis tool.

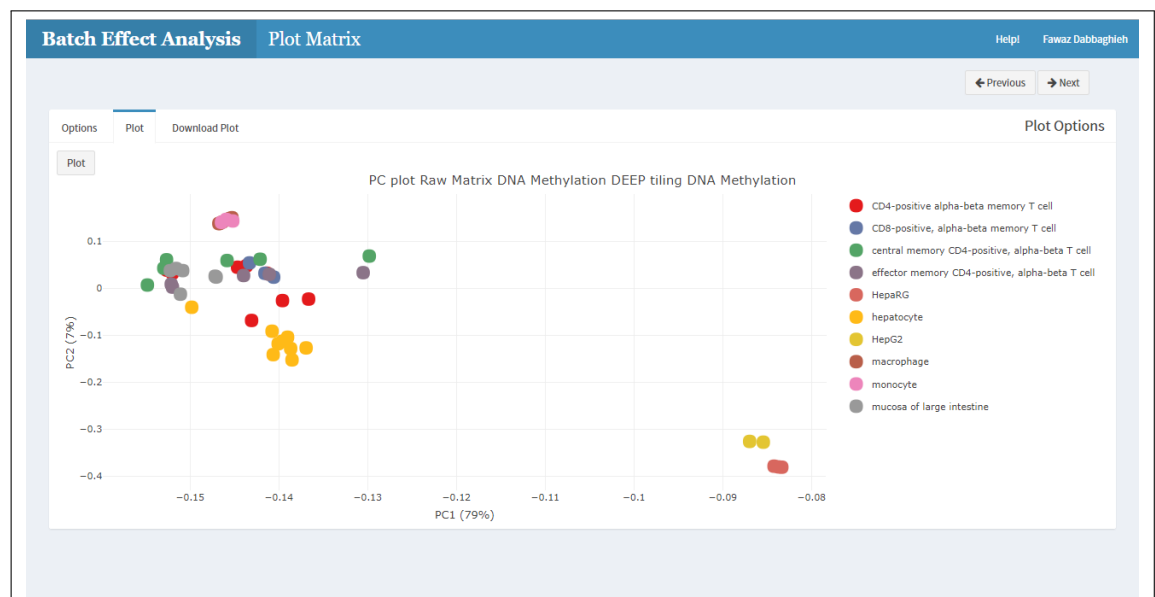


Figure 5.16: Batch analysis tool interactive PCA chart: it contains the experiments values based on the computed score matrix values. Coloring the experiments based on any metadata field is possible. In this figure, they are colored according to their BioSource (*biosource_name* field).

Batch Effect Analysis | Batch Effects | Help! | Fawaz Dabbaghieh

← Previous | Next →

Choose method to calculate scoring matrix

- ☐ ComBat
- ☐ SVA
- ☐ Supervised SVA
- ☒ RUV

Select Outliers (Optional)

Batch Corrected Matrices

ruv-0.005-5

Adjust matrix

Regularization parameter for the unwanted variation (nu.coeff)

0.00001

Method to estimate House Keeping Genes

- ☒ Estimate using rank analysis
- ☐ Use a predefined list

Quantile probability used in the rank analysis estimation [0,1]

0.005

Desired rank for the estimated unwanted variation term

5

Download Data

Show 10 entries

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max
S1_HF03_BICM_CT_WGBS_S_1_MCSv3.20150424.GRCh37.cpg.filtered.CG.bedgraph	-2.29499	-0.18142	-0.13790	-0.09275	-0.05260	0.70851
41_HF14_LIHe_St_WGBS_S_1_MCSv3.20151127.GRCh37.cpg.filtered.CG.bedgraph	-0.9291	0.0765	0.1833	0.1477	0.2458	0.9732
41_Hm08_LIHe_St_WGBS_S_1_MCSv3.20151210.GRCh37.cpg.filtered.CG.bedgraph	-0.6231	0.2182	0.3670	0.3170	0.4537	1.9835
41_HF05_LIHe_St_WGBS_S_1_MCSv3.20150424.GRCh37.cpg.filtered.CG.bedgraph	-1.88101	-0.00086	0.08301	0.05812	0.14013	1.10691

Figure 5.17: Configuring and executing the batch effect correction. Different methods are available: ComBat, SVA, Supervised SVA and RUV. The tool *help* page describes each method. In this interface, users can define the parameters of the method and remove outliers before performing the batch effect correction.



Figure 5.18: Batch correction result comparison. This interface allows users to compare the corrected to the original results in an interactive *PCA* chart. The initial values are presented on the left side and the corrected are on the right side.

to the data for reproducibility and efficiency. Moreover, maintaining a controlled vocabulary can help test newly generated metadata before it is submitted or added to *DeepBlue*; all these factors combined provide a flexible and reliable strategy that can easily be implemented at the beginning of a project, which makes sure that any metadata file added is up-to-date and the occurrence of mistakes are minimized. This strategy was successfully applied within *DEEP* and can be extended to other projects.

The tool for analyzing and correcting batch effects tackles a common issue in multi-institutional consortia. It provides a straightforward and intuitive interface for users to analyze and visualize the batch effect in the data, and correct the values before downstream analysis.

Finally, this use case presented a complete data and metadata analysis and correction tools. First with a metadata cleaning pipeline and then with the interactive tool for analyzing and correcting batch effect on epigenomic data. Both tools use the *DeepBlueR* package for accessing and processing (epi)genomic data, showing the power of the combination of the *R/Bioconductor* environment with the *DeepBlue Server*.

5.4 Conclusion

Public data portals enable researchers to gain access to terabytes of epigenomic data. This large amount of available data creates a strong demand for data analysis in statistical environments such as *R*, but it is not effective on local computers due to the volume of the data. The *DeepBlueR Bioconductor/R* package enables *R* users to tap directly into the *DeepBlue Server* to operate on large epigenomic datasets. *DeepBlue* transparently transforms the results to *R* data structures that are directly used with *R/Bioconductor* packages for visualization (*Gviz*), statistical learning (*glmnet*) or data analysis (*ComBat*). *DeepBlueR* contains additional examples and documentation in its vignette package⁶. Finally, as demonstrated in dozen examples and the four complete use cases, *DeepBlueR* is a powerful tool to study the epigenomics complexities, having the support of the data and operations available in the *DeepBlue Server* in the powerful *R/Bioconductor* environment.

⁶ <https://bioconductor.org/packages/release/bioc/html/DeepBlueR.html>

6

Accessing DeepBlue data from the web-browser

This chapter describes the DeepBlue Web Portal that is a companion tool for the DeepBlue Server. This tool was implemented by the author with support of Obaro Odiete.

Accessing public epigenomic data is a cumbersome task: users need to access different data portals, manually search for the experiments data files through different metadata formats, download the files, and then filtering the data from these files for the desired genomic regions. Efforts such as the IHEC data portal (Bujold *et al.* 2016) provide a unified data portal containing the IHEC members' data, which however lacks features such as complete metadata search, downloading all data on a specific epigenomic mark or sample, or downloading subsets of the regions contained in the selected data files.

In contrast, the *DeepBlue Server* API provides a rich set of operations for finding, listing, retrieving, and analyzing epigenomic data, but it is better suited for researchers with programming knowledge. The *DeepBlue Web Portal*¹ was developed for filling the gap between the powerful *DeepBlue Server* API and researchers without a background in programming. The *DeepBlue Web Portal* allows for searching and downloading the (epi)genomic data available in the *DeepBlue Server* using a web interface.

The *DeepBlue Web Portal* is composed of visual components like grids, data tables, full-text search interface, and download wizards for facilitating the full usage of the data available in the *DeepBlue Server*. It serves as a companion tool of the *DeepBlue Server*, where researchers can find and select the data to be used in the *DeepBlue Server* or to be downloaded and analyzed directly.

6.1 Accessing public epigenomic data

Accessing the public epigenomic data is a cumbersome task: users need to access different data portals, such as, *DEEP*², *BLUEPRINT*³, *ENCODE*⁴, *ROADMAP*⁵, *CEEHRC*⁶,

¹ <http://deepblue.mpi-inf.mpg.de>

² <http://deep.dkfz.de>

³ <http://dcc.blueprint-epigenome.eu>

⁴ <https://www.encodeproject.org>

⁵ <http://www.roadmapepigenomics.org/data>

⁶ <http://epigenomesportal.ca/edcc/index.html>

for finding the necessary data for their analysis. In this process, users must visit many web pages, search manually through the web page containing the metadata in different formats, using the infamous *CTRL+F* command, download files without knowing their content a priori, extract the relevant data, and finally, filter and manipulate the data.

The process of finding the desired epigenomic data is divided into the following steps: (i) access epigenomic data portals; (ii) manually search lists of metadata files; (iii) find the files that match the required metadata; (iv) download the files (usually *gigabytes* of data); (v) check if the files contain the desired data; (vi) filter and manipulate parts of the files required for the analysis.

The *IHEC* data portal⁷ improves this situation by providing a centralized repository with data and semi-unified metadata. Here, we say *semi-unified* rather than *unified* because many projects do not follow the *IHEC* metadata specification⁸. Furthermore, the *IHEC* data portal provides files for download, but it is not possible to select and download a collection of files, or only parts of specific files.

Due to the increasing amount of available epigenomic data, a more accessible, and faster way of accessing the epigenomic data is necessary; otherwise, it will not be possible to perform large-scale epigenomic data analysis in a reasonable amount of time and resources, both computational and human.

6.2 DeepBlue Web Portal

The *DeepBlue Web Portal* was implemented with the goal of streamlining access to the public epigenomic data provided by the *DeepBlue Server* and gives users an overview of the (epi)genomic data available in the *DeepBlue Server*. It also allows them to perform simple data selection and filtering for further download. It provides a web interface to a subset of the *DeepBlue Server* API operations, focusing on the activities for finding, searching, and downloading the data.

This web portal is a companion tool of the *DeepBlue Server* and its API, providing visual components like grids, data tables, full-text search, and download wizards for facilitating the full usage of the epigenomic data available in the *DeepBlue Server*.

Similar to the *DeepBlue Server*, the *DeepBlue Web Portal* supports both anonymous and controlled access via a simple user registration. Anonymous users have access to all public data, and to all operations for retrieving the data. Registered users, depending on their permission privileges, can upload, curate, or remove data. In addition, the user interface supports an interactive learning approach by automatically showing a tutorial-style guide to first-time users. The principal functionalities of the *DeepBlue Web Portal* are presented and described in the following sub-sections.

Dashboard

When accessing the *DeepBlue Web Portal*, the first interface is a data dashboard which gives an overview of the available data (Figure 6.1). It provides charts showing how the data is distributed among the projects, genomes, biosources, epigenetic marks, and

⁷ <http://epigenomesportal.ca/ihec/>

⁸ <https://github.com/IHEC/ihec-metadata>

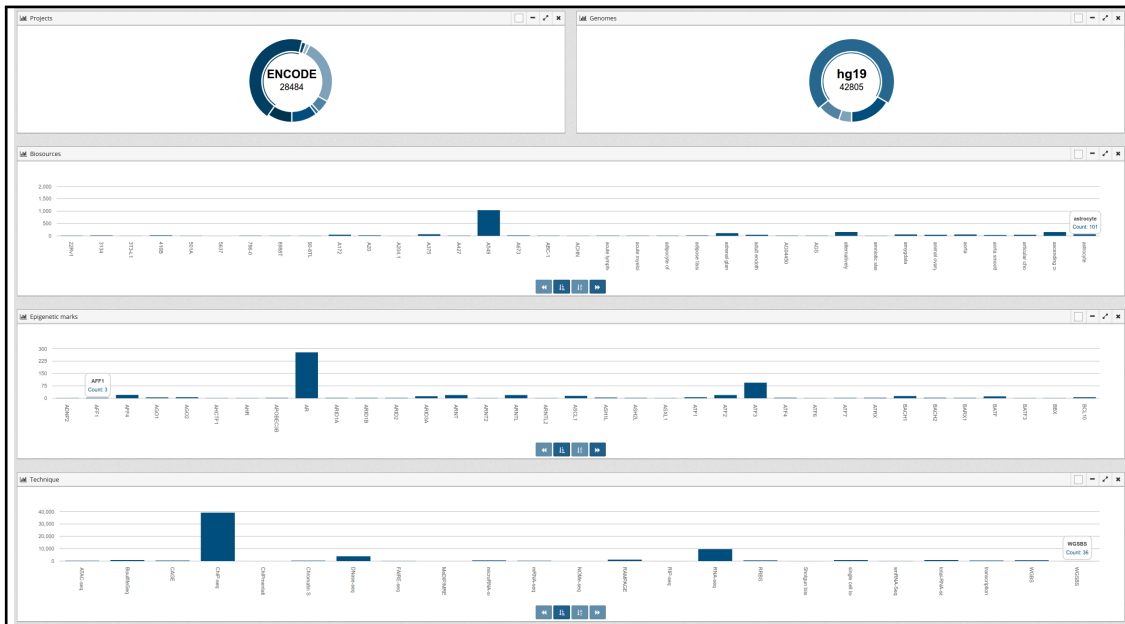


Figure 6.1: The dashboard provides an overview on the available data, segmenting it by its metadata content.

techniques. Besides visualizing the data distribution, users can also use these charts for selecting and downloading the data.

Experiments Grid

The experiments grid allow users to visualize all data available. It supports filtering by the experiments metadata, e.g., genome, epigenetic mark, and biosource. Figure 6.2 shows the grid interface, displaying all *ChIP-seq* peak data from the BLUEPRINT project. Besides visualizing the available data, the experiments grid provides a data export functionality, where users can select experiments and export the list of chosen experiments to a programming script in the *R* or *Python* programming languages. By using this functionality, user benefit from selecting the data in an easy interface and performing data analysis in their favorite programming language. Section 6.3.1 presents a use case that exemplifies this functionality.

Full-text search

The web portal provides a convenient full-text search interface through which users can search epigenomic, annotation, or metadata records similarly to performing a web search in "google". The web full-text search usage is simple: users type the query words that must be in the target content description: metadata for experiments or annotations, or fields of the metadata entities. Extra querying elements can be included, for example, double quotes to form query terms with more than one word, such as "*DNA methylation*" or "*ChIP-seq*". A plus signal (+) is used in front of a word that must be contained in the content, and a minus signal (-) is used to define a word that must not appear in the results content. In addition, the queries are case insensitive. For example, Figure 6.3 shows the

[illegible]

Figure 6.2: The grid interface provides a visualization of all available data in a simplified grid format. This example shows all *ChIP-seq* peak data from the BLUEPRINT project.

query "DNA methylation" +BLOOD -coverage +GRCH38⁹ on all experiments that contain DNA methylation, blood, and GRCh38 in their metadata, but not coverage. Internally, this web interface uses the *DeepBlue Server search* operation, where all requests are handled on the server in a few seconds.

Entities info

The entities info interface is a simple interface that envelopes the *info* operation. This interface answers simple questions about any *DeepBlue Server* content, where users can obtain the full information of any *DeepBlue ID*, from metadata fields to processed requests.

Experiments data table

The experiments data table provides a straightforward way of obtaining information and accessing the data and metadata of the experiment stored in the *DeepBlue Server*. It is a data table that shows the experiments' mandatory metadata fields as individual columns, contains an extra column with the *extra-metadata* content, and a button for easy access to preview the data. Users can use the text input in the header of each column for quickly filtering the table content. Figure 6.4 shows the filtering process: users type the values and an auto-complete *ComboBox* is displayed to assist the user in finding the right metadata field. The filtered result is shown instantly, such that users can quickly explore the data provided by the *DeepBlue Server*. This interface is also the entry point for downloading experiments data via the web portal. The use case in Section 6.3.2 explores the data downloading process using this interface.

Experiments ▾ "DNA Methylation" +BLOOD -coverage +GRCh38

★ **e94988 - S013RUA1.CPG methylation calls.bs call.GRCh38.20160531.wig**
 ● experiment ● GRCh38 ● dna methylation ● peripheral blood mononuclear cell ● bisulfite-seq ● BLUEPRINT Epigenome
Sample Info
BIOMATERIAL_PROVIDER : Prof.dr. E. Vellenga, University Medical Centre Groningen - Department of Hematology
BIOMATERIAL_TYPE : Primary Cell
CELL_TYPE ... [More](#)

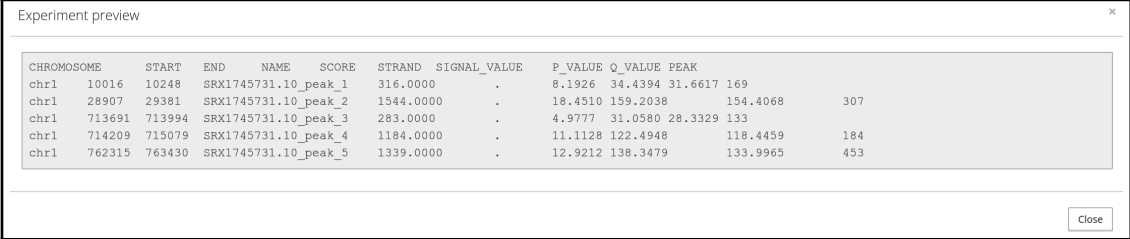
★ **e94992 - S013RUA1.CPG methylation calls.bs cov.GRCh38.20160531.wig**
 ● experiment ● GRCh38 ● dna methylation ● peripheral blood mononuclear cell ● bisulfite-seq ● BLUEPRINT Epigenome
Sample Info
BIOMATERIAL_PROVIDER : Prof.dr. E. Vellenga, University Medical Centre Groningen - Department of Hematology
BIOMATERIAL_TYPE : Primary Cell
CELL_TYPE ... [More](#)

★ **e111090 - S013RUA1.hyper meth.bs call.GRCh38.20160531.bed**
 ● experiment ● GRCh38 ● dna methylation ● peripheral blood mononuclear cell ● bisulfite-seq ● BLUEPRINT Epigenome
Sample Info
BIOMATERIAL_PROVIDER : Prof.dr. E. Vellenga, University Medical Centre Groningen - Department of Hematology
BIOMATERIAL_TYPE : Primary Cell
CELL_TYPE ... [More](#)

Figure 6.3: The full-text interface provides a straightforward way for finding (epi)genomic data and metadata content stored in the *DeepBlue Server*. This figure shows the query "DNA methylation" +BLOOD -coverage +GRCh38 on all experiments that contain DNA methylation, blood, and GRCh38 in their metadata, but not coverage.

ID	Experiment	sign	Description	Genom	h3k	endomet	Sampl	Technique	Projec	Meta data	Preview
ID	Name	Type	Description	Genome	h3k36me3 h3k4ac h3k4me1 h3k4me2 h3k4me3 h3k9me1 h3k9me2	stoma endometrium epithelium	Sample	Technique	Project	Metadata	Preview
e107190	NCCHD_ChipSeq_EM02065C_H3K27ac_20140724_R1.rmdup.wig	signal		GRCh38			s11740	ChIP-seq	CREST	-- View metadata --	Q
e107206	NCCHD_ChipSeq_EM10085C_H3K36me3_20140724_R1.rmdup.wig	signal		GRCh38			s11747	ChIP-seq	CREST	-- View metadata --	Q
e107216	NCCHD_ChipSeq_EM02065C_H3K27me3_20140724_R1.rmdup.wig	signal		GRCh38			s11740	ChIP-seq	CREST	-- View metadata --	Q
e107222	NCCHD_ChipSeq_EM1018EC_H3K4me3_20150615_R1.rmdup.wig	signal		GRCh38	H3K4me3	endometrium epithelium	s11744	ChIP-seq	CREST	-- View metadata --	Q
e107250	NCCHD_ChipSeq_EM1016EC_H3K36me3_20150615_R1.rmdup.wig	signal		GRCh38	H3K36me3	endometrium epithelium	s11756	ChIP-seq	CREST	-- View metadata --	Q
e107259	NCCHD_ChipSeq_EM10085C_H3K9me3_20140724_R1.rmdup.wig	signal		GRCh38	H3K9me3	endometrial stroma	s11747	ChIP-seq	CREST	-- View metadata --	Q
e107264	NCCHD_ChipSeq_EM1016EC_H3K9me3_20150615_R1.rmdup.wig	signal		GRCh38	H3K9me3	endometrium epithelium	s11756	ChIP-seq	CREST	-- View metadata --	Q
e107296	NCCHD_ChipSeq_EM02065C_H3K9me3_20140724_R1.rmdup.wig	signal		GRCh38	H3K9me3	endometrial stroma	s11740	ChIP-seq	CREST	-- View metadata --	Q
e107326	NCCHD_ChipSeq_EM10085C_H3K4me1_20140724_R1.rmdup.wig	signal		GRCh38	H3K4me1	endometrial stroma	s11747	ChIP-seq	CREST	-- View metadata --	Q

Figure 6.4: The experiments data table provides a convenient access to all *DeepBlue Server* experiments with their metadata. This figure shows filtering the experiments by its epigenetic mark.



CHROMOSOME	START	END	NAME	SCORE	STRAND	SIGNAL_VALUE	P_VALUE	Q_VALUE	PEAK
chr1	10016	10248	SRX1745731.10_peak_1	316.0000	.	8.1926	34.4394	31.6617	169
chr1	28907	29381	SRX1745731.10_peak_2	1544.0000	.	18.4510	159.2038	154.4068	307
chr1	713691	713994	SRX1745731.10_peak_3	283.0000	.	4.9777	31.0580	28.3329	133
chr1	714209	715079	SRX1745731.10_peak_4	1184.0000	.	11.1128	122.4948	118.4459	184
chr1	762315	763430	SRX1745731.10_peak_5	1339.0000	.	12.9212	138.3479	133.9965	453

Figure 6.5: The experiments data table provides a data preview functionality. This figure shows the preview of an experiment data and its columns.

The experiments data table also provides a data *preview* functionality where users can visualize the first five lines of an experiment data instantly (Figure 6.5). In this way, users do not need to download the data for seeing their content, but can instead just check the content of any data file with a single click. This functionality uses the *DeepBlue Server preview_experiment* operation.

Auxiliary data listing

The *DeepBlue Web Portal* also provides similar data tables for auxiliary data like annotations, metadata entities, column types, and genes. By these data tables, users can easily verify if some annotation or metadata field is available in the *DeepBlue Server*, as well as obtain detailed information on various entities, e.g., the description of a column type.

BioSources hierarchy

The BioSources hierarchy interface provides visualization to the BioSources hierarchy provided by the *DeepBlue Server*. The interface shows a data tree where users can visualize and select experimental data. First, users can explore the nodes of the tree, each representing a BioSource term. Then, users can select the BioSource nodes for choosing their respective samples. When selecting a node, all its inner nodes are also selected, and a list of available samples is displayed on the right-hand of the web interface (Figure 6.6). Samples can be selected and unselected, and their respective experiments are displayed in a data table, which can be used for downloading their data.

Access to previous processing requests

An interface is provided for accessing to the previous processing requests. This interface contains a data table where registered users can access all their previous requests. It is possible to obtain information about the request date and to download its processed data. Due to privacy reasons, this functionality is not available to anonymous users because a user could see what other users are processing in the *DeepBlue Server*.

Data insertion

The data insertion has two interfaces: one for uploading new annotations and another for inserting new column types without requiring the use of programming scripts. Both interfaces are simple to use, requesting only a minimum amount of information about

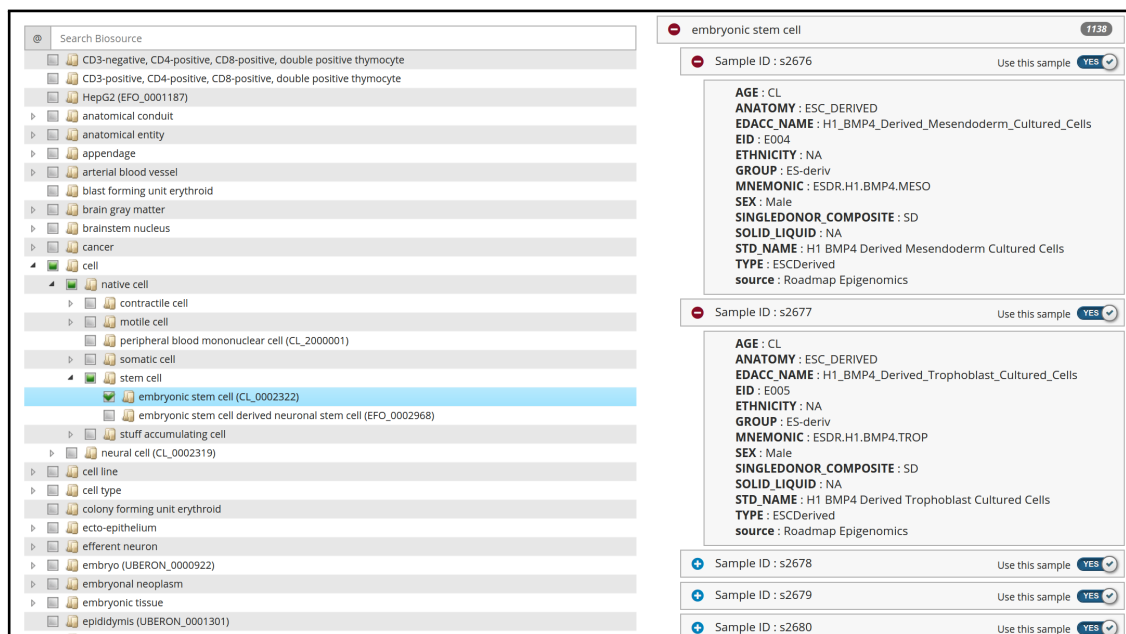


Figure 6.6: The BioSources hierarchy interface allows users to visualize the relationship between BioSources. This figure shows the selection of *ESC* and the samples that are annotated with this BioSource.

the data to be included. These interfaces were developed as an experiment to verify if users prefer to use a web interface or scripting programs to upload their data into *DeepBlue Server*. The conclusion was that due to the large number of annotation datasets, scripting was the best option. In any case, these interfaces are still available for users with permission for inserting data into the *DeepBlue Server*.

Data curation

The data curation interface provides a straightforward way of correcting the metadata content of a set of experiments. This functionality requires that the user is registered and has permission for changing and inserting data into the *DeepBlue Server*. When accessing this interface, users select a set of experiments by their metadata content. In the next step, users can modify the selected experiment metadata, changing its content, including or removing *extra-metadata* fields, and altering the column types. The interface uses the *clone_dataset* operation for storing the changes on the server. This operation does not change or duplicate the original experiment data, but creates another experiment, with the new metadata content that references the original data. In this way, the original data is neither lost nor unnecessarily duplicated.

Data removal

The data removal interface provides an easy way for privileged and privileged users to remove data from the *DeepBlue Server*. It uses the *remove* operation, allowing to remove any entity by its *ID*.

Documentation & Users' feedback

The two last interfaces are the documentation and users' feedback. The documentation web interface is a hub to all available *DeepBlue Epigenomic Data Server* ecosystem documentation, including API reference, examples, use cases, and user manual. The users' feedback is a direct link to the *User Echo* service¹⁰. Using this service, users can write comments, feedback, and bugs reports¹¹. In short, the users' feedback is a channel where users can communicate with the developers and help improving the *DeepBlue Epigenomic Data Server* ecosystem.

6.3 Usage examples

Here, two use cases are detailed demonstrating the usefulness of the *DeepBlue Web Portal*: (i) using the grid interface for selecting and exporting data to a python script; (ii) selecting and downloading specific regions from multiple experiments.

6.3.1 Visually selecting and exporting data to a Python script

For supporting users with the cumbersome task of finding and downloading the necessary epigenomic data, this use case demonstrates how to perform such task using the grid interface. Here, we explain how the experiments for the *DeepBlue Server* use case *Summarize genes expression from hepatocytes experiments* (Section 4.6.5) are found.

The first step is to open the *Experiments grid* interface in the web portal. There, select the option *signal* in *Data type*, *CREST* in *Projects*, *GRCh38* in *Genome*, *mRNA* in *Epigenetic Marks*, *Hepatocyte* in *BioSources*, and *RNA-seq* in *Techniques* (Figure 6.8). The grid contains only one cell (Figure 6.8), holding the number 8, representing the eight experiments that match these criteria.

The following steps are executed in the grid interface for obtaining a source code in *R* or *Python* containing the selected experiments name:

- Click on the exhibited grid cell for selecting its experiments.
- Scroll the interface to its bottom, where the data table with the selected experiments is located (Figure 6.9).
- Click on the button *Export data*, displaying the generated source code in *R*.
- Click on the *Python* text in the top for displaying the *Python* code (Figure 6.10).
- Click on the button *Copy to Clipboard* for copying its content, then, paste it in the *Python script* with the data analysis script.

Finally, this use case and the use case presented in Section 4.6.5 present an integrated use of the web interface together with a programmatic analysis. This demonstrates how users can have the best of both worlds: an intuitive interface for searching and finding the required data, and a powerful API for epigenomic data manipulation.

¹⁰ <https://deepblue.userecho.com/>

¹¹ It is encouraged to report bugs in the *DeepBlue's* GitHub repository: <https://github.com/MPIIComputationalEpigenetics/DeepBlue>

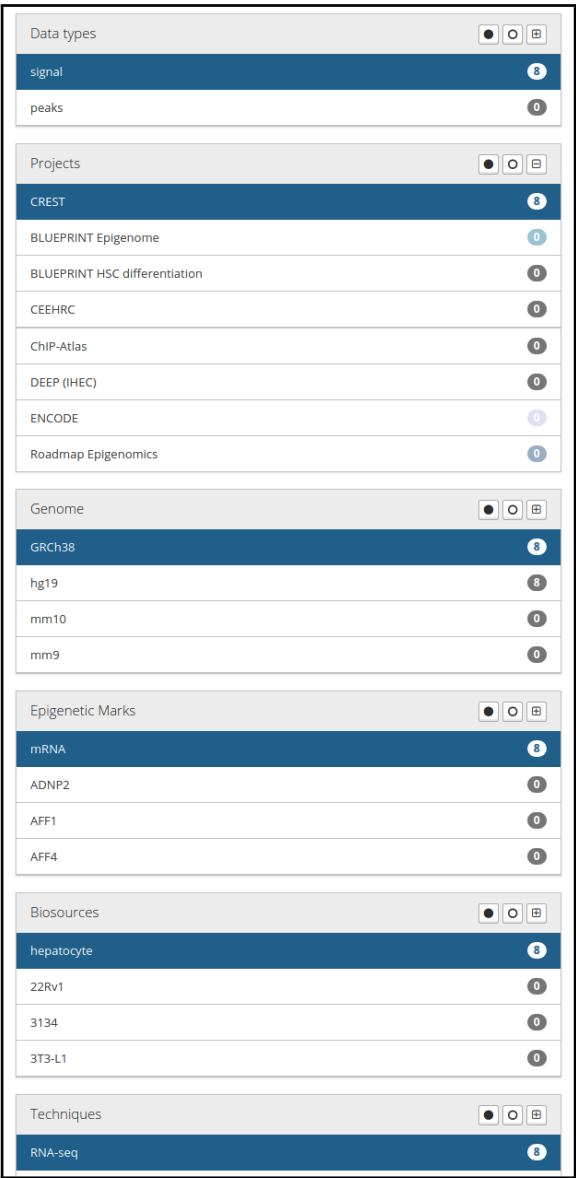


Figure 6.7: DeepBlue Web Portal Grid: This figure shows the filtering options applied to the data for selecting DNA methylation data from CREST for further analysis.



Figure 6.8: DeepBlue Web Portal Grid: This figure shows the filtering result where the grid contains a single cell displaying the eight filtered hepatocytes experiments.

ID	Experiment Name	Type	Description	Genome	Epigenetic Mark	Biosource	Sample	Technique	Project
e107269	HPC27_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11737	RNA-seq	CREST
e107278	HPC17_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11734	RNA-seq	CREST
e107279	HPC8_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11735	RNA-seq	CREST
e107281	HPC25_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11731	RNA-seq	CREST
e107283	HPC20_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11736	RNA-seq	CREST
e107285	HPC28_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11733	RNA-seq	CREST
e107347	HPC35_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11730	RNA-seq	CREST
e107350	HPC6_hg38_RNAseq.ucsc.bedgraph	signal		GRCh38	mRNA	hepatocyte	s11732	RNA-seq	CREST

Figure 6.9: DeepBlue Web Portal Grid: selected experiments data in the grid are displayed in the data table.

```

R Python

grid_experiments = [ "HPC6_hg38_RNAseq.ucsc.bedgraph",
                     "HPC35_hg38_RNAseq.ucsc.bedgraph",
                     "HPC25_hg38_RNAseq.ucsc.bedgraph",
                     "HPC28_hg38_RNAseq.ucsc.bedgraph",
                     "HPC20_hg38_RNAseq.ucsc.bedgraph",
                     "HPC8_hg38_RNAseq.ucsc.bedgraph",
                     "HPC17_hg38_RNAseq.ucsc.bedgraph",
                     "HPC27_hg38_RNAseq.ucsc.bedgraph" ]

```

Figure 6.10: DeepBlue Web Portal Grid: export selected data to *Python*.

6.3.2 Selecting and downloading specific regions from multiple experiments

Retrieving epigenomic data from multiple experiments is a complex task, especially when only a subset of this data is required, e.g., regions overlapping a specific annotation. Currently, epigenomic data portals only provide the download of individual files, where users must download files individually, manually merge, and then select the regions of interest.

The *DeepBlue Web Portal* provides an easy way of downloading the required epigenomic data. The download process takes three steps and all have visual guidance: (i) selecting experiments, (ii) selecting columns, chromosomes, and overlapping regions; (iii) requesting and downloading the regions.

This use case demonstrates how to use the *Experiments data table* interface (Section 6.2) for selecting regions from *ChIP-seq* of *precursor B cell* experiments that overlap with CpG islands:

- Select the experiments by filtering the data table with the following metadata attributes: *peaks* as *Type*, *H3K27ac* as *Epigenetic mark*, *precursor B cell* as *Biosource*, and *ChIP-seq* as *technique* (Figure 6.11).
- After the data table filtering, 13 experiments are left.
- Select these experiments by double-clicking each row.
- Scroll to the bottom of the web page where the list of selected experiments is displayed at the bottom of the web page. If necessary, an experiment can be deselected by double-clicking its row.

ID	Name	Type	Description	Genome	Epigenetic Mark	Biosource	Sample	Technique	Project
e91844	S01GTBH1.ERX1304156.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10406	chip-seq	BLUEPRINT Epigenome
e91833	S01GSDH1.ERX1304206.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10404	chip-seq	BLUEPRINT Epigenome
e91820	S01GRFH1.ERX1304204.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10402	chip-seq	BLUEPRINT Epigenome
e91809	S01GQHH1.ERX1304138.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10400	chip-seq	BLUEPRINT Epigenome
e91802	S017E3H1.ERX1345474.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10397	chip-seq	BLUEPRINT Epigenome
e91785	S0179DH1.ERX1345471.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10395	chip-seq	BLUEPRINT Epigenome
e91773	S0177HH1.ERX1347893.H3K27ac.bwa.GRCh38.20160527.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10393	chip-seq	BLUEPRINT Epigenome
e91762	S0176JH1.ERX1304201.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10392	chip-seq	BLUEPRINT Epigenome
e94917	S0174NH1.ERX1302279.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s11152	chip-seq	BLUEPRINT Epigenome
e95010	S0175LH1.ERX1304135.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s11153	chip-seq	BLUEPRINT Epigenome

Showing 1 to 10 of 13 entries (filtered from 10 total entries)

Selected experiments
 Double click the row to unselect an experiment. It will be removed from the data table.
 For downloading the data, click on the Download button in the end of the page. You will be redirected to the download page.

Selected experiment(s)

Q

ID	Experiment	Type	Description	Genome	Epigenetic mark	Biosource	Sample	Technique	Project
ID	Experiment Name	Type	Description	Genome	Epigenetic Mark	Biosource	Sample	Technique	Project
e91762	S0176JH1.ERX1304201.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10392	chip-seq	BLUEPRINT Epigenome
e91773	S0177HH1.ERX1347893.H3K27ac.bwa.GRCh38.20160527.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10393	chip-seq	BLUEPRINT Epigenome
e91785	S0179DH1.ERX1345471.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10395	chip-seq	BLUEPRINT Epigenome
e91802	S017E3H1.ERX1345474.H3K27ac.bwa.GRCh38.20160310.bed	peaks		GRCh38	h3k27ac	precursor B cell	s10397	chip-seq	BLUEPRINT Epigenome

Figure 6.11: DeepBlue Web Portal: selecting data with the experiments data table.

- Click on the button *Proceed to the download page* for proceeding to the next data selection and filtering step.

The *Download experiments* interface (Figure 6.12) allows users to select the content of the data that is retrieved. It also allows to select the experiments columns and to annotate regions with meta-fields. In the presented use case, the sequence of commands are the following:

- Select the columns *CHROMOSOME*, *START*, *END*, *NAME*, *P_VALUE*, and *Q_VALUE*.
- Annotate the regions with the meta-field *@SAMPLE_ID* for identifying the source of each region.
- Filter the regions by overlapping with CpG island. If necessary, it is possible to select regions by their chromosome or genomic location.
- Click the button *Request Download*. This button triggers the processing in the *DeepBlue Server* and redirect the user to the *Request Status* interface.

The *DeepBlue Server* processes the request on demand. This means that users must wait for the processing to finish before they can download the data. The request generated with the use case presented above takes about eight seconds to process. Figure 6.13 shows the request processing status interface, which provides information such request *ID*, the selected data, start, and end time. When processing the request is finished, the user must click on the button *Download* for downloading the generated data file. It is possible to access the selected data directly in *R* or *Python* using the *get_request_data* operation, or *deepblue_download_request_data* command in *DeepBlueR*, with its request *ID* (r3444577).

Common Column(s)

CHROMOSOMESTARTENDNAMEP_VALUEQ_VALUE

Usage: Use the dropdown to include a column. Click on the X to exclude the column

Optional Column(s)

Usage: Use the dropdown to include a column. Click on the X to exclude the column

Meta Column(s)

@SAMPLE_ID

Usage: Use the dropdown to include a column. Click on the X to exclude the column

Calculated Column(s)

Usage: Use the dropdown to include a column. Click on the X to exclude the column

Genomic Coordinate

Genomic Coordinate

Chromosome

Usage: Keep empty to select all chromosomes. Use the dropdown to include a chromosome. Click on the X to

Overlapping with Annotations

Overlapping

Annotations

Cpg Islands

Usage: Use the dropdown to include an annotation. Click on the X to exclude the annotation

Figure 6.12: DeepBlue Web Portal Grid: selecting data columns and regions.

ID	r3444577
Request Info	command : get_regions format : CHROMOSOME, START, END, NAME, P_VALUE, Q_VALUE, @SAMPLE_ID
Request Details	intersect (query q7665190): <ul style="list-style-type: none">experiment_select (query q7665189):<ul style="list-style-type: none">experiment name: S01GTBH1.ERX1304156.H3K27ac.bwa.GRCh38.20160310.bed, S01GSDH1.ERX1304206.H3K27ac.bwa.GRCh38.20160310.bed, S01GRFH1.ERX1304204.H3K27ac.bwa.GRCh38.20160310.bed, S017E3H1.ERX1345474.H3K27ac.bwa.GRCh38.20160310.bed, S0179DH1.ERX1345471.H3K27ac.bwa.GRCh38.20160310.bed, S0177HH1.ERX1347893.H3K27ac.bwa.GRCh38.20160527.bed, S0176JH1.ERX1304201.H3K27ac.bwa.GRCh38.20160310.bed, S0174NH1.ERX1302279.H3K27ac.bwa.GRCh38.20160310.bed, S0175LH1.ERX1304135.H3K27ac.bwa.GRCh38.20160310.bed, S017B9H1.ERX1304154.H3K27ac.bwa.GRCh38.20160310.bed, S017C7H1.ERX1302280.H3K27ac.bwa.GRCh38.20160324.bed, S017D5H1.ERX1304137.H3K27ac.bwa.GRCh38.20160310.bed, S01GQHH1.ERX1304138.H3K27ac.bwa.GRCh38.20160310.bedannotation_select (query q987491):<ul style="list-style-type: none">annotation: CpG Islands
Start Time	2018-Oct-22 22:16:31
End Time	2018-Oct-22 22:16:39
State	ready
<div>Download</div>	

Figure 6.13: DeepBlue Web Portal request info interface displays information and allows users to download the data of a request ID.

The presented use case demonstrates how simple it is to use *DeepBlue Web Portal* for selecting and filtering epigenomic data without the need of downloading a set of experiments and manually filtering them. It showed that the data selection is performed by a few clicks and then processed in the *DeepBlue Server*, providing precisely the required data for further analysis.

6.4 Conclusion

The amount of available epigenomic data is steadily increasing, but as presented in Section 6.1, methods for accessing these data follow the old paradigm of the *one file, one experiment, one download*. They are forcing users to search in many different data portals for the required data, and to download dozens of files for analyzing a small portion of them.

The *DeepBlue Web Portal* provides an elegant interface by which users can search, list, and efficiently download epigenomic data. Besides the data, the *Web Portal* provides facilities for accessing *DeepBlue* metadata entities and also to curate experiments metadata.

Overall, the *DeepBlue Web Portal* is a convenient interface and companion tool for the *DeepBlue Server*, being used by hundreds of users. The *Web Portal* is a powerful tool, but it does not provide all operations needed for complex epigenomic data analysis using an web interface. Offering more sophisticated ways for complex epigenomic data analysis is the domain of the tool *DIVE*, which is presented next in Chapter 7.



Large scale interactive analysis of epigenomes



DIVE is a web application compatible with all modern web browsers. It was developed by the author. The use case in the Sections 7.5.1 and 7.5.2 were developed with support of Dr. Shinya Oki. The use case in the Section 7.5.3 was developed during the author's visit in the Professor Dr. Mikita Suyama laboratory.

Large scale epigenomic data analysis has to deal with three fundamental difficulties: locating the necessary experiments, extracting the data of interest from these experiments, and analyzing the data. The *DeepBlue Server* facilitates obtaining the required data, and also provides some basic analysis capabilities such as gene and overlapping enrichment methods (Section 4.5.7). The *DeepBlue Web Portal* simplifies finding the desired data set, but it lacks data analysis functionalities. Furthermore, retrieving and analyzing hundreds of different files is not efficiently possible using a standard desktop computer. For handling these issues, and consequently, improving large-scale (epi)genomic data analysis, the web tool *DIVE* was developed.

This chapter presents *DIVE*, a web tool for large-scale epigenomic data analysis. While users need to have knowledge in programming when using the *DeepBlue* API (Section 4.5), and the *Web Portal* (Chapter 6) provides a subset of the *DeepBlue* API, *DIVE* aims to provide a comprehensive, dynamic, and visual interface for analyzing large-scale epigenomic data.

DIVE is a dynamic analysis web application that uses the *DeepBlue Server* metadata, data, and API without requiring the user to have any programming skills. Users can

perform analysis in hundreds of epigenomic data files in seconds with a few mouse clicks.

7.1 Web tools for analyzing epigenomic data

Web tools for analyzing (epi)genomic data are being used routinely. Based on the use mode, these tools can be classified in four groups: (i) genome browsers: UCSC Genome Browser (Kent *et al.* 2002) and Ensembl Genome Browser (X. M. Fernández and Birney 2010); (ii) workflow managers: Galaxy (Giardine, Riemer, Hardison, Burhans, Elnitski, Shah, Y. Zhang, Blankenberg, Albert, Taylor, *et al.* 2005b; Giardine, Riemer, Hardison, Burhans, Elnitski, Shah, Y. Zhang, Blankenberg, Albert, Taylor, *et al.* 2005a; Goecks *et al.* 2010; Blankenberg *et al.* 2010); (iii) machine learning and enrichment: EpiGraph (Bock *et al.* 2009) and EpiAnnotator (Pageaud *et al.* 2018); (iv) whole genome analysis: EpiExplorer (Halachev *et al.* 2012).

Genome Browsers are “web-based application for displaying genomic annotations and other features” (Stein *et al.* 2002). Genome Browsers are *de facto* tools for inspecting genomic annotations, like epigenetic marks, CpG islands, and genes. Their major drawback is that users must manually screen along the genome, inspecting the genomic locations and annotations individually, which requires comprehensive knowledge about the research question at hand to interpret the presented information. In short, these tools strongly rely on human interaction and data interpretation, which does not scale with the growing amount of experiment files.

Workflow managers are software that connects different data processing and analysis tools to form a data processing and analysis pipeline. Workflow managers provide connectors to existing software, providing links between different software packages, and configuring their interlinked their execution. The two most frequently used workflow managers in bioinformatics are Taverna (Wolstencroft *et al.* 2013), a desktop application, and Galaxy, a command line and web application. Both tools provide access to dozens of different data sources and software tool repositories. However, users must (i) manually find the data from various sources; (ii) verify the data quality; (iii) build the data workflow. These tools do not require any programming skills, but users must know the required data and required software tools for constructing the data processing and analysis workflows. Furthermore, workflow managers usually scale by distributed computing, the scalability of individual workflow components is less of an issue. In this way, workflow managers retrieve dozen to thousands of data files, for examining only a fraction of their content.

EpiExplorer is one of the first tools for whole (epi)genome analysis. It is a point-and-click web application that allows users to inspect epigenomic region sets by overlapping them with genomic annotation, e.g., with CpG islands or known TFBSs, thus, obtaining insights about this region set. As opposed to *Genome Browsers*, *EpiExplorer* compares the region set to the whole genome and exhibits the annotations with their number of overlaps. *EpiExplorer* provides dozens of pre-processed datasets (histone modifications, chromatin accessibility, DNA methylation, CSS, TFBS, CpG island, lamina-associated domains, conservation, repeat elements, and Genes and annotations). Due to its ease of use, *EpiExplorer* was well received in the community, having thousands of users. The

main issues with *EpiExplorer* are the need for pre-processing the region set, which can take several hours to complete and also limits the query flexibility for the users.

The machine learning and enrichment analysis methods are mathematical methods for obtaining insights on the (epi)genomic data in a semi-automatized way. *EpiGraph* (Bock *et al.* 2009) is one of the first web tools that enabled users to use machine learning methods for epigenomic data enrichment (Section 3.4). In *EpiGraph*, users must define the features and parameters for calculating the enrichment, a task that is complex for biologists and bioinformaticians because it requires knowledge in statistics and in machine learning. Another tool, *EpiAnnotator* (Pageaud *et al.* 2018) provides a more straightforward user interface, where users can enrich genomic overlapping regions with existing datasets, using the *LOLA* (Sheffield and Bock 2016) method, but besides its ease of use, its functionality is limited to enrichment analysis.

As previously presented, the *DeepBlue Server* provides mechanisms for data selection and operation, where users can build whole-genome (epi)genomic data processing workflows directly in the data server. It also provides operations for performing overlapping enrichment analysis (Section 4.5.7). One of the main advantages of using *DeepBlue* is the ability to process hundreds of epigenomic data files simultaneously. It enables users to explore, analyze, and obtain insights from large-scale datasets, not only regarding the number of experiments but also regarding the variety, obtaining data from many different samples and epigenetic marks. *DeepBlue* allows for comparing a region set with all its imported datasets, similarly to what *EpiExplorer*, *LOLA*, and *EpiAnnotator* perform. But users must develop scripts using the *DeepBlue* API, which renders analysis prohibitive for users without any programming skills.

DIVE was developed for supporting users in performing large-scale epigenome analysis without the need to implement code. *DIVE* empowers scientists by providing a complete visual interface where they can obtain insights as well as to formulate a scientific hypothesis on their epigenomic data.

7.2 DIVE - Diving in the epigenomic data

DIVE is an intuitive web-based epigenomic data analysis tool with the primary objective to support researchers in analysis of large-scale epigenomic datasets. *DIVE*'s main functionality is similar of *EpiExplorer*: a set of genomic regions is compared regarding overlap with existing datasets, and the results are visualized as bar charts. But *DIVE* can perform these comparisons dynamically, without the need of pre-processing the data. Thus, the tool provides flexibility regarding the queries that users can visually build with only a few mouse clicks.

The region set that the user wants to analyze is called *query* in *DIVE*. For the analysis, users select additional region sets as the so-called *comparison* sets. The *query* and *comparison* region sets can be compared to all datasets available in *DIVE* (in the following: *DIVE datasets*). Figure 7.1 shows the main *DIVE* interface, pointing out the elements, *query*, *comparison* region-sets, and *DIVE region-set*. The *query* regions can be filtered by their length, content, or by overlapping with a given DNA sequence motif. The filtering steps results are displayed (Figure 7.1-B). The result from a filtered step can also be used as a *comparison* region-set, allowing users to select which regions are used for comparison.

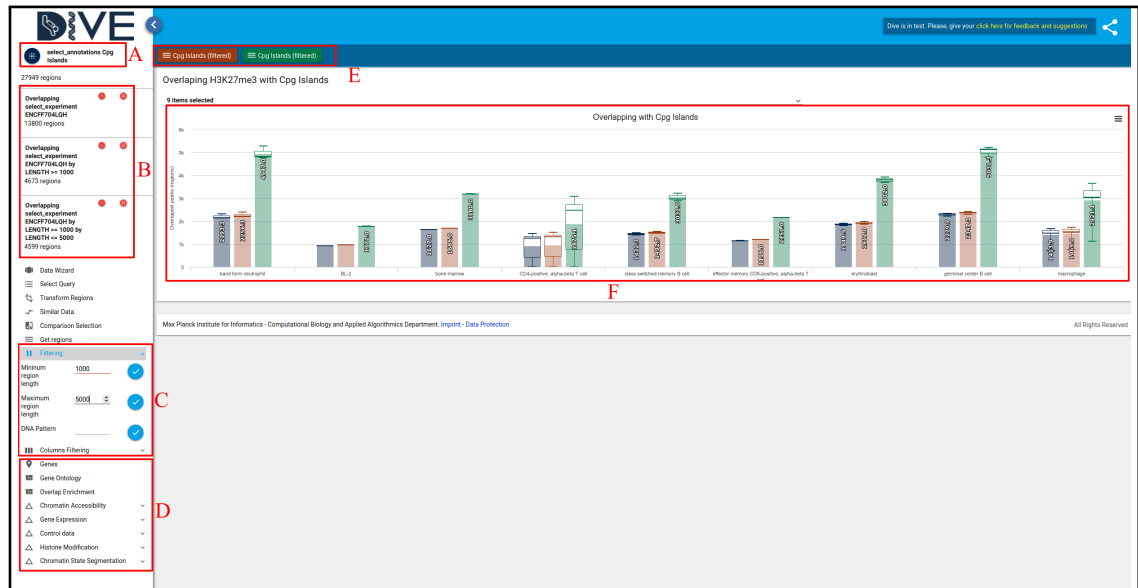


Figure 7.1: DIVE interface main elements: (A): the *query* region-set that is being analyzed, in this case; (B): list of the filters that are being applied to the *query* region-set; (C) filtering options based on region lengths and overlapping by DNA pattern that can be applied to the *query* region-set; (D) access to the genes enrichment analysis, gene ontology enrichment analysis, and overlapping enrichment analysis, also access to the datasets available for overlapping comparison (Chromatin Accessibility, Gene Expression, Control Data, Histone Modification, and Chromatin State Segmentation); (E) List of the *comparison* region-sets, where their color corresponds to the colors in the charts; (F) bar charts with the counts of overlaps of the *query* and *comparison* region-sets to the selected region sets (H3K27me3 experiments), where each group represents a distinct BioSource.

7.3 Main features

DIVE presents several innovations to web-based epigenomic large-scale data analysis: ability to analyze a large amount of data in a few seconds, guided data analysis, flexible queries processed at run-time, dynamic region-sets, multiple region-sets source, and enrichment analysis. *DIVE*'s main analysis functionalities are detailed in the following sections.

Full visual interface

DIVE is a visual and intuitive web application. Users do not need knowledge in programming, and all operations are performed in *DIVE*'s graphical interface, with instantaneous feedback to the user. Its visual interface is built using *User Interface* (UI) elements like wizards¹, data tables, dynamic menus, and charts displaying the analyzed data, for users visualizing and analyzing their data.

¹ <https://uxplanet.org/wizard-design-pattern-8c86e14f2a38>

Large amount of data ready to use

All public data in DeepBlue is automatically available for analysis. Although *DIVE* works only with region-based datasets, this nevertheless means that, at the time of writing, more than 35,000 experiments from six reference genomes and more than 2,000 biosources can be analyzed. Moreover, dozens of genomic annotation datasets and five gene models are available to the user, providing diverse starting points for in-depth epigenomic data analysis. Furthermore, *DIVE* can operate on multiple samples from the same *BioSource*, improving the quality of the analysis as compared to the single-sample approach implemented in *EpiExplorer*.

Guided data analysis

Due to the substantial number of datasets available in *DIVE*, the tool guides users through the analysis, helping them to choose the datasets for data comparison and enrichment. When users access *DIVE*, the tool initializes a wizard guiding them through the process of selecting the genome, annotations, and data. During this process, *DIVE* automatically suggests the most similar and dissimilar BioSources to potentially broaden the scope of the intended analysis. At the end of this guided setup, all parameters for the analysis are available and the analysis run can be started with a single mouse click. If necessary, these parameters can be easily changed during the data analysis.

When selecting a similar or dissimilar BioSources, *DIVE* also suggests the inclusion of its sub-terms, based on the ontologies, in the data analysis (Figure 7.2).

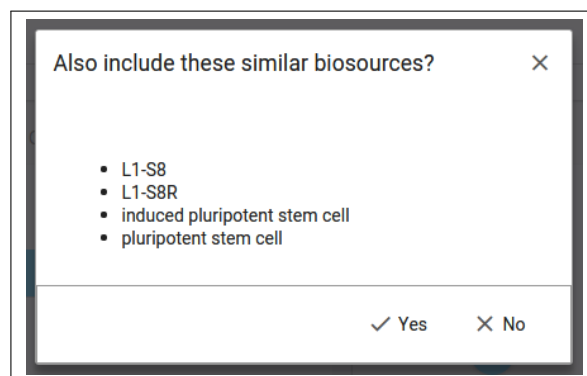


Figure 7.2: *DIVE* suggesting sub-terms when selecting a BioSource: clicking on the *pluripotent stem cell* term, *DIVE* shows a menu asking if it must include the sub-terms for automatically selection in the overlapping interfaces.

Run-time data processing

DIVE converts information provided by the user to *DeepBlue API* operations, which are processed, and the results are presented in the form of charts and tables. To this end, *DIVE* uses the *DeepBlue Server* for processing queries at run-time, without the need of pre-processing data. In short, users can upload their datasets and start analyzing the data immediately. In case of an analysis request that cannot be processed instantaneously, the interface is updated with preliminary results. Besides, all operation results are permanently stored to speed up future requests that use the same data again.

Flexible queries

Since all queries are processed at run-time, users can build and execute flexible queries. Besides the usual functionality of filtering regions by overlapping regions from other region-sets, it is also possible to filter regions based on their content, genomic location, overlap with DNA sequence patterns, genes, or genes annotated with specific *GO* terms. The queries also have adjustable parameters, for example, the filtering by overlap with a DNA sequence pattern is not limited to pre-defined patterns but can apply any valid regular expression content.

Furthermore, it is possible to build on a query using its result as input for another query. Hence, *DIVE* allows user to build queries chain for probing the epigenomic data with more intricate research hypotheses.

Dynamic region-sets

DIVE provides thousands of datasets but, in addition, users can generate new region-sets from existing datasets. For example, it is possible to compute flanking regions from gene regions, such that researchers can define promoter regions as appropriate for their analysis. It is also possible to generate several derived region sets and compare the results. Besides flanking regions, it is possible to create tiling regions with configurable length, or to select gene sets based on *GO* annotations. It is also possible to transform a region-set by extending its regions.

DIVE also generates *CSS* region-sets dynamically. Usually, the experimental data is organized by samples, where each file contains the whole-genome *CSS* per sample. To facilitate the analysis, *DIVE* organizes the region sets by their chromatin states. This is realized by finding all distinct chromatin states in all data files (Section 4.5.6 and subsequent filtering of regions by their chromatin state (Section 4.5.5). In this way, *DIVE* generates on-the-fly new region-sets based on the existing chromatin states.

Multiple region-sets sources

Different sources can be used for providing the *query* and *comparison* region-sets. These input sources are:

- Experiments and Annotations from DeepBlue can be directly accessed in *DIVE*. The only limitations are that the experiments data must be *peaks* and also public.
- Tiling regions with configurable length can be generated at run-time.
- DNA sequence motif can be used for generating a region-set on the fly that is composed of genomic regions matching the specified DNA sequence motif.
- DeepBlue's query *ID* can be used to load the region-set referenced by the given query *ID*.
- Genes and *GO* terms can be used to build a region-set from the genes selected by their names, *ID*, or annotated by *GO* terms.
- User files can be uploaded to be analyzed in *DIVE*.

As highlighted above, *DIVE* supports many data sources, also allowing users to use region-sets from previous analyses performed with the *DeepBlue* API. The opposite direction is also possible, i.e., user can filter and analyze a region-set in *DIVE*, and later access this region-set in a programming environment using the *DeepBlue* API.

Enrichment analysis

DIVE provides a simple overlap analysis using genes and also three types of enrichment analyses: (i) *GO*; (ii) *LOLA* (Section 4.5.7); (iii) fast overlapping enrichment analysis (Section 4.5.7).

The genes enrichment analysis is a straightforward method where the *query* and *comparison* region-sets are overlapped with the gene models provided by *DeepBlue*, and the number of overlapped genes is returned to the user. With this functionality, researchers can observe how many of their regions overlap with genes and regions around the genes. It is also possible to generate region-sets based on the gene locations, for example, promoter region-sets based on different values for the distance to the gene *TSS* and its regions length.

The *GO* enrichment analysis is similar to the gene enrichment, but a second step counts the *GO* terms that annotate the overlapped genes. The results are displayed in a bar plot, containing the options for filtering the *GO* terms by three properties: (i) minimum number of overlapping genes with this *GO* term; (ii) minimum fraction of overlapping genes in relation to the total number of genes annotated with the same *GO* term; (iii) *GO* term overlaps *P-value*. Users can easily modify these three filtering attributes to focus on more specific *GO* terms. Figure 7.3 illustrates a use case of performing the *GO* enrichment analysis on a CpG island region-set. It is possible to filter the query to the overlapping genes annotated with a specific *GO* term by simply clicking on the *GO* bar.



Figure 7.3: *DIVE*: Gene Ontology Enrichment: this image shows the result of a *GO* enrichment analysis followed by manual filtering the results through three parameters: minimum number of overlapping genes with the *GO* term; minimum ratio of overlapping genes with the total number of genes; *GO* term overlaps *P-value*. All these parameters can be modified and the results are exhibited instantly, which in this case, four *GO* terms that 69 to 95 genes annotated with these terms overlap with the *query* regions.

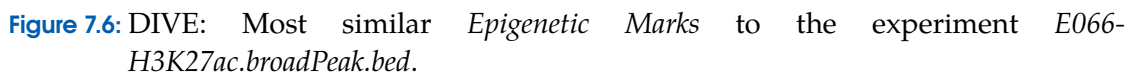
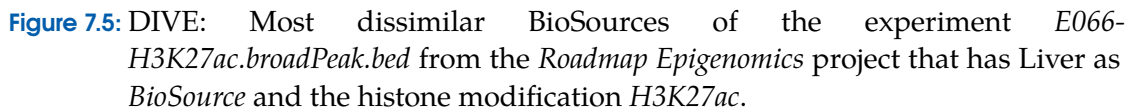
The *LOLA* enrichment analysis uses the *DeepBlue* API operation *enrich_regions_overlap* (Section 4.5.7). *DIVE* provides a wizard interface, which guides the user through the process of selecting the background data and comparison datasets. The user can select data from all previously listed input sources, including all *peaks* experiments available in the *DeepBlue Server*. Figure 7.4 exhibits the enrichment result of the CpG island against

Source	Scenario	Scenario ID	Accession	Accession ID	Sequence name	Seq. length	Seq. ID	Accession	Seq. length	Seq. ID	Accession	Seq. length	Seq. ID	Accession	Seq. length	Seq. ID	Accession	Seq. length	Seq. ID		
nr78784	endothelial cell of rat	Chr10:Stemata	S00L8M41	12 12 1	27054	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	80,4095	14542	518	8	280362	180	217	172	132,667	
nr77720	endothelial cell of rat	Chr10:Stemata	S00C5H41	12 12 8	12230	12	Active TSS	H3k4	Stand	H3K4Me1	H3K27Ac	0	139,6108	11304	1749	13135	28731	379	30	379	136,667
nr79356	CD4-positive, alpha1	Chr10:Stemata	C00Z7WH1	12 12 1	23466	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	99,1408	12914	3544	12525	22836	245	166	245	137,333	
nr78628	CD4-positive, alpha1	Chr10:Stemata	S00L8M41	12 12 12	13719	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	111,9655	11414	2271	13025	28363	371	82	371	151,333	
nr78978	endothelial cell of rat	Chr10:Stemata	S00L8M42	12 12 1	25700	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	73,8579	13631	4793	10608	280487	211	252	252	154,667	
nr79070	revisited cell	Chr10:Stemata	S00D9H41	12 12 8	27245	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	65,1141	14099	6698	9360	27872	164	287	287	155,333	
nr79128	CD4-positive, alpha1	Chr10:Stemata	C00D9H41	12 12 1	24444	11	Active TSS	H3k4	Stand	H3K4Me1	H3K27Ac	0	139,6108	11304	1749	13135	28731	379	30	379	136,667
nr79158	CD4-positive, alpha1	Chr10:Stemata	S00C2TH1	12 12 1	2013	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	85,5714	12378	1070	12378	26578	164	186	186	157,333	
nr79160	CD4-positive, alpha1	Chr10:Stemata	S00B9TH1	12 12 8	18099	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	89,8922	12024	3043	12415	28437	315	161	315	159	
nr78822	endothelial cell of rat	Chr10:Stemata	S00L8M41	12 12 8	19199	12	Active TSS	H3k4	Stand	H3K4Me1	H3K27Ac	0	126,5342	10802	1776	13637	283704	435	50	435	162
nr79006	alternative activate	Chr10:Stemata	S00K2TH1	12 12 23	23317	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	78,7828	12784	3942	11655	281556	256	230	256	162,333	
nr79930	endothelial cell of rat	Chr10:Stemata	S01G1CH1	12 12 8	26060	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	55,998	15341	8345	9009	277135	158	338	338	166,667	
nr78600	endothelial cell of rat	Chr10:Stemata	S01G1CH1	12 12 8	28926	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	65,2136	13753	5525	10046	279955	204	296	296	166	
nr78602	endothelial cell of rat	Chr10:Stemata	S00Z9TH1	12 12 8	25877	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	67,9609	13414	5021	11025	280487	204	283	283	167,333	
nr79746	CD4-positive, alpha1	Chr10:Stemata	C00Z2TH1	12 12 1	18471	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	64,818	12916	3796	12916	27969	206	279	279	166,333	
nr79782	CD4-positive, alpha1	Chr10:Stemata	S00Z7TH4	12 12 8	16536	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	56,4492	12016	2605	12043	283575	384	342	342	169,333	
nr77044	macrophage	Chr10:Stemata	S00D9TH1	12 12 8	26736	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	64,482	13684	5254	10795	28054	384	302	302	170,333	
nr79784	effector memory CD	Chr10:Stemata	C00J3TH1	12 12 8	19599	12	Active TSS	H3k4	Stand	H3K4Me1	H3K27Ac	0	107,8358	10930	2126	13099	283354	415	95	415	170,333
nr79188	CD4-positive, alpha1	Chr10:Stemata	S00W6H1	12 12 8	21250	11	Active TSS	H3k4	Stand	H3K4Me1	H3K4Me1	79,1783	12373	3640	1206						

The fast overlapping enrichment analysis uses the *enrich_regions_overlap* operation (Section 7.5). This method used by *DIVE* is explained in Section 7.4.3. In short, this method is used to list the most similar and dissimilar *BioSources* during the data selection wizard. Figure 7.5 shows the *BioSources* most dissimilar from the experiment file *E066-H3K27ac.broadPeak.bed* from the *Roadmap Epigenomics* project that has Liver as *BioSource* and the histone modification *H3K27ac*. The goal of this interface is to guide users through the most similar and dissimilar *BioSources*, helping them to find and focus on the putatively informative experiments to be used as *comparison* region-sets and the *DIVE* *region-sets* to be compared to.

Sharing analysis results

DIVE allows users to share the analysis with colleagues by sharing a *Uniform Resource Locator* (URL) address. Any collaborator can open the shared URL and the *query* and *comparison* datasets, including all the previously applied filtering steps.



DIVE empowers researchers to analyze a large amount of (epi)genetic using a simple point-and-click interface. *DIVE* uses operations from the *DeepBlue Server* and improved methods of accessing these operations to realize its analysis capabilities. In this section, the three most important methods are explained: (i) regions overlap analysis, (ii) utilizing and displaying multiple samples per *BioSource*, (iii) computing experiments data similarity, which uses the fast enrichment analysis processing.

7.4.1 Overlapping regions

The overlap analysis is performed using the *DeepBlue* operation *overlap* for each pair of *query* or *comparison* and *DIVE* region-set. To emphasize the efficiency of *DIVE* together

with *DeepBlue*, one can consider an exemplary case of a single query and two comparison datasets being overlapped with 40 *DIVE* region-sets. In this scenario, a total of 120 (3×40) *DeepBlue* workflows have to be processed, each consisting of at least two data selections and one overlap operation. Ideally, all these operations have to complete within fractions of a second to provide the user with essentially instantaneous analysis results.

The previous example contains a small number of region-sets. In *DIVE*, it is usual for users to select 25 *BioSources*, where each one has approximately six samples, totaling in 150 *DIVE* region-sets. Still, if each of these 150 region-sets can be processed in a matter of seconds, the total amount of time surpasses the range of minutes, which is a long time for waiting. So, even though the *DeepBlue* Server algorithms are optimized, in the current implementation of *DIVE* and *DeepBlue* Server, it is not feasible to process all 450 overlapping computations in real time. Furthermore, even if the processing time could be reduced, the amount of available data is growing continuously, which easily outweighs any performance gains on the algorithmic level.

For these reasons, *DIVE* implements a paradigm called *process-and-display* that displays the overlapping results as soon they are available. The *process-and-display* implementation consists of three different components: *DIVE*, middleware, and *DeepBlue* Server: (i) *DIVE* sends a request to the middleware to trigger the overlap computation. It provides the *Query ID* and a list of experiment *IDs*, and, if necessary, a list of filters that must be applied to the experiment's data; (ii) the middleware executes the operation *overlap* in the *DeepBlue* Server for each pair of *query* or *comparison* region-set and datasets; (iii) the middleware monitors the processing using the operation *info* and retrieves the results when the workflow is finished. (iv) Concurrently, *DIVE* keeps probing the middleware for new results, receiving the partial results, and continuously updates the overlapping chart.

The *process-and-display* method allows users to visualize the results of large-scale datasets while they are still being processed, which avoids the undesirable "idling" stage for the user. The same method is used for finding the similar *Epigenetic Marks* (Figure 7.6) and *BioSources* (Figure 7.5), and it can be extended to other *DIVE* components.

7.4.2 Sample aggregation

As an improvement over previous tools, *DIVE* has built-in support for analyzing multiple samples per *BioSource*. As shown in Figure 7.1, *DIVE* groups experiments by their *BioSource*. Each *query* or *comparison* region-set and *BioSource* bar is composed of one or more samples, and consequently, one or more experiment data files. The results of these multiple experiments are summarized in a bar chart enriched with a box plot.

Figure 7.7 shows parts of the *DIVE* interface, focusing on a bar chart with three bars enriched with a box plot. For constructing this chart, each pair formed by a *query* or *comparison* region-set and a *sample* is processed by the *DeepBlue* Server using the operation *overlap*. *DIVE* first groups the results by *BioSource*, and then uses these results for calculating the average, median, and the first and third quartile values for each *BioSource*. The average is used for the bar chart while the other values, together with the minimum and maximum, are used in the box plot. Users can select individual samples for performing overlap operations by clicking on these charts. In this way, users can obtain an overview of the overlap results of the selected *BioSources*, as well as identify potential outliers.

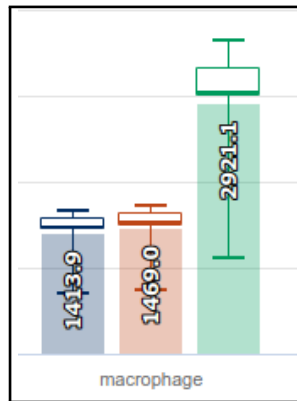


Figure 7.7: DIVE: Multiple samples per *BioSource*: this chart contains three bar charts enriched with a box plot. Showing the average overlapping for each *BioSource* and the samples with the lowest and highest number of overlaps, the first and third quartile, and the median.

7.4.3 Finding similar experiments

Listing similar *BioSources* and *Epigenetic Marks* for a given region-set is one of the most exciting features in *DIVE*. This method enriches the *query* region-set using all available experiments annotated with the same *Genome*. In this setting, the use of the original *LOLA* method is infeasible due to its substantial memory consumption and comparatively slow processing speed. For this reason, the *DeepBlue Server* implements an approximate and faster version of this method in the operation *enrich_regions_fast* (Section 4.5.7). This operation can compute regions overlap enrichment of a region-set containing thousands of regions against a set of thousands of datasets in a matter of minutes rather than hours or even days when using the original *LOLA* method.

The *enrich_regions_fast* operation is fast, but it still requires seconds to minutes for its complete processing of a region-set containing thousands of regions against thirty thousands of datasets². Due to this reason, it also uses the *process-and-display* approach. Using this approach, rather than executing the operation *enrich_regions_fast* once with all available experiments, many executions of *enrich_regions_fast* are performed, where the experiments are separated by *BioSources* or *Epigenetic Marks*, choosing the one that provides the lower granularity. It means that *DIVE* aims to executing more *enrich_regions_fast* operation with a smaller number of experiments in each of these executions. This choice exploits the *DeepBlue Server* parallelism, allowing to execute multiple workflows simultaneously and obtaining new results faster, hence, continually updating the *UI* with new results as soon as they are received by *DIVE*.

Processing the enrichment is the first step of finding the similar *BioSources* and *Epigenetic Marks*. The second step is to use the enrichment results for calculating the similarity rank of an experiment. The following steps summarize the rank calculation process:

1. Obtain the enrichment results (partial or all)
2. Sort results by the average of the p-value, odds-ratio, and support ranks.³
3. Select the top *N*% results

² number of peak experiment datasets annotated with the genome *hg19* in the *DeepBlue Server*

³ Inverting the sort order returns the most dissimilar values.

4. Group experiments by *BioSource* and *Epigenetic Mark*
5. Calculate the average scores for each *BioSource* and *Epigenetic Mark* group
6. Sort the group scores
7. Return the scores to the *UI* and display them to the user

These operations are executed repeatedly after receiving new results and are processed in the web browser, immediately updating the *UI* with the results charts. Furthermore, users can cancel the processing, or change the parameters; for example, the cut-off value used in the third step for selecting the top $n\%$ results. When this parameter is modified, the calculation is executed in the user web browser, displaying the new results immediately.

7.5 Use cases

To illustrate the usefulness of *DIVE* for analyzing epigenomic data, here presents three typical use cases that until now had to be performed programmatically and manually in epigenetic studies: (i) analyzing the enrichment of chromatin states for finding important transcription factors in cell pluripotency; (ii) analyzing transcription factors by overlapping with chromatin states; (iii) comparing data from different consortia.

7.5.1 Analyzing the enrichment of chromatin states for finding important transcription factors in cell pluripotency

TFs have important roles in the regulation of gene expression. This use case aims at discovering which TFs preferentially co-locate with chromatin states representing strong enhancer regions.

The use case starts by utilizing the visual interface for visualizing *similar data*. This interface gives an overview of the most similar epigenetic marks and BioSources regarding the query region set. In the next step, the enrichment analysis provided by the *LOLA* method is used for analyzing the enrichment of the active chromatin state regions using *TF* experiments. Using the enrichment results, the most similar and dissimilar TFs, comparing to the query region set, are selected for performing an overlap analysis using all selected chromatin states region-sets for double-checking the findings.

For this purpose, this use-case uses the BioSource *H1-hESC*, which is a cell line derived from *HSCs*, and select four chromatin states as region-sets: *Strong Enhancers* (4)⁴ as main *query* region-set, and *Strong Enhancer* (5)⁵, *Active Promoters* (1), and *Poised Enhancers* (3) as *comparison* region-set. After selecting the region-sets, *DIVE* performs a fast enrichment analysis for finding similar experiments (Section 7.4.3) for obtaining the similar epigenetic marks and BioSources.

The *Similar epigenetic marks* chart (Figure 7.8) shows the epigenetic marks with peaks most strongly colocalizing with the *Strong Enhancer* (4) from a *H1-hSC* sample. The result supports with the biological literature, with *DNA accessibility* being the most prominent

⁴ These number represents the ENCODE chromatin states codes: http://rohshdb.cmb.usc.edu/GBshape/cgi-bin/hgTables?db=hg19&hgta_group=regulation&hgta_track=wgEncodeBroadHmm&hgta_table=wgEncodeBroadHmmNheKHM&hgta_doSchema=describe+table+schema

⁵ The states *Strong Enhancer* (4) and *Strong Enhancer* (5) are different states

epigenetic information, with other marks and TFs responsible for by controlling the gene expression and development, such as the TF *NIPBL*, which plays an important role in human limbs development (Muto *et al.* 2014), *H3K9/14ac*, and the TF *CTCF*.

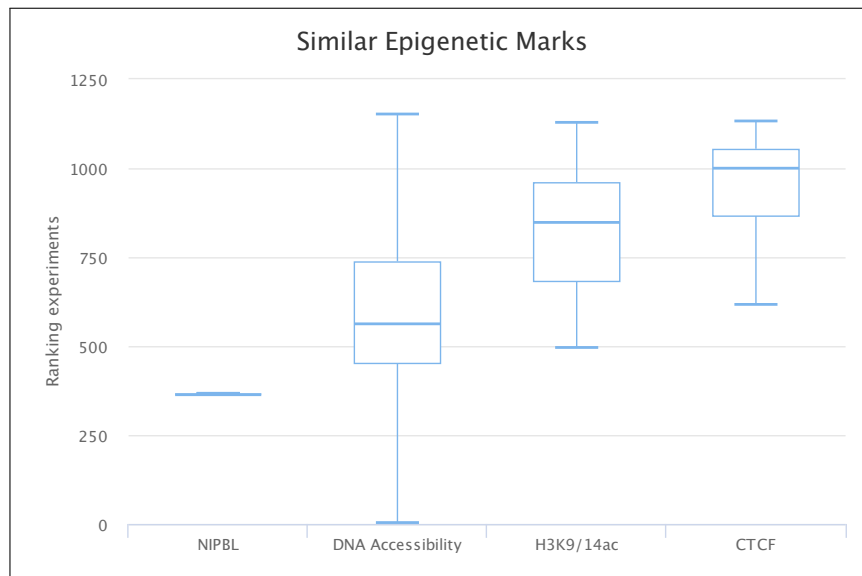


Figure 7.8: Overview on the Strong Enhancer regions of the *h1-hSC*: this chart shows the epigenetic marks with peaks most strongly colocalizing with the *Strong Enhancer* (4) from a *h1-hSC* sample. The result supports the biological literature, with *DNA accessibility* being the most prominent epigenetic information, with other marks and TFs responsible for by controlling the gene expression and development, such as the TF *NIPBL*, *H3K9/14ac*, and the TF *CTCF*.

The fast enrichment analysis for obtaining experiments (Section 7.4.3) is also used for obtaining the most similar BioSources. The three most similar BioSources regarding the *query* region-set are: *L1-S8R*, which is an induced pluripotent stem cell line, *NT2/D1* derived from lung cancer, and pluripotent stem cell that this analysis uses.

The following step consists of performing an enrichment analysis using TFs. TFs from *ENCODE* and *Chip-Atlas* are selected and their enrichment is analyzed in the *query* region-sets (*Strong Enhancers* (4) regions). From this analysis, sorting the results by their *Odds Ratio*, the highest ranked TF is *POU5F1* that is specifically expressed in *ESC* and necessary for pluripotency (G. Shi and Jin 2010). This protein acts together with *SOX2* and *NANOG* (Boyer *et al.* 2005) that are top ranked in the list of TFs.

The lowest ranked TF is *EZH2*, which is a TFs involved in transcriptional repression via methylation of 'Lys-9' (*H3K9me*) and 'Lys-27' (*H3K27me*) of histone *H3* (Kirmizis *et al.* 2004), leading to transcriptional repression of the affected target gene.

It is also interesting that the TF *FOXP1* is the second lowest ranked TF. It is interesting because *FOXP1* may act as a tumor suppressor (Koon *et al.* 2007), which tumor suppressors are usually repressed in *HSC* because they inhibit regenerative capacity by promoting cell death or senescence in stem cells (Pardal *et al.* 2005).

The previous enrichment analysis yielded the most common TFs that preferentially co-locate with active sites in the genome. The next step is to analyze the previously

selected three states (active, strong enhancer, and poised). First, these states are compared with the *TF POU5F1* from the BioSources *pluripotent stem cell* and *H1-hESC* (Figure 7.9). This figure shows that the *strong enhancer* state overlaps with the largest number of *POU5F1* regions and the repressive *poised* state overlaps with the smallest number, which is in line with the current biological knowledge (Boyer *et al.* 2005; X. Chen *et al.* 2008).

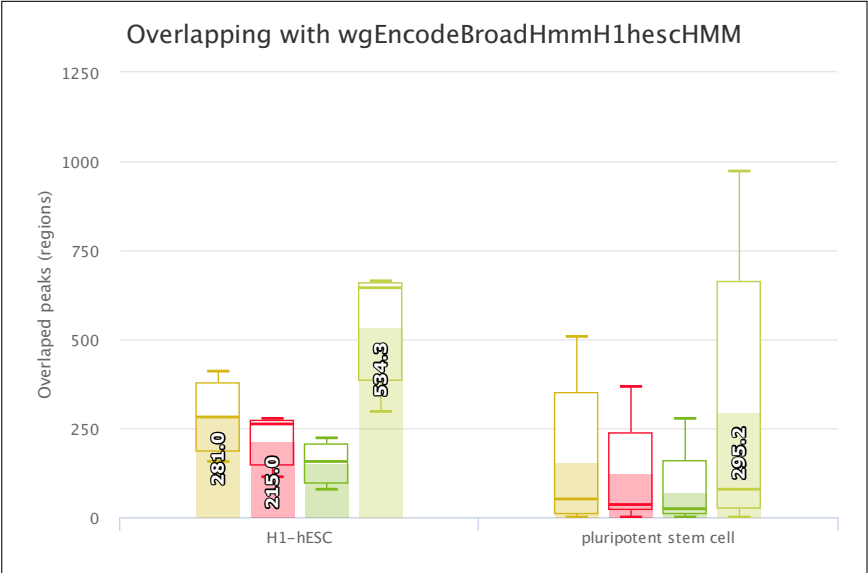


Figure 7.9: Overlapping to the chromatin states of *POU5F1* of experiment SRX1053369: this chart shows that the *Strong Enhancers* (4 and 5) state, both in yellow color, overlaps with the largest number of *POU5F1* regions and the repressive *poised* state overlaps with the smallest number, aligning to the current biological knowledge.

This use case demonstrates how simple it is to perform an enrichment analysis in *DIVE* for finding important TFs for the pluripotency in *HSC*, such as *POU5F1*, *SOX2*, and *NANOG*. For a deeper understanding, the next use case focuses on analyzing individual experiment data from these TFs.

7.5.2 Analyzing transcription factors by overlapping with chromatin states

This use case aims at visualizing the differences between stem cell BioSources and other BioSources regarding chromatin states and TFs partially responsible for pluripotency.

To this end, we select experiments from *POU5F1*, *NANOG*, and *SOX2*, which are TFs specifically expressed in stem cells and necessary for the pluripotency (Boyer *et al.* 2005). In addition, data from *EZH2*, which is a *TF* involved in transcriptional repression (Kirmizis *et al.* 2004), and *EP300* that is broadly activating in enhancers (Arbel *et al.* 2019), is selected. Table 7.1 shows the selected TFs with the respective experiment, the *TF* function, and the color which the *TF* is presented in the following charts. The following charts presents the TFs bars following the order that they are listed in the following table.

The analysis consists of overlapping the selected experiments with the chromatin state *Strong Enhancer* of the BioSource fibroblast of lung, *GM12878*, *H1-hESC*, *HepG2*, *HMEC*,

Transcription Factor	Experiment Name	Function	Bar Color
POU5F1	SRX1053369	pluripotency factor	Green
EZH2	SRX317590	transcriptor repressor factor	Red
EP300	SRX027482	broadly activated in enhancers	Purple
NANOG	SRX702041	pluripotency factor	Green
SOX2	SRX512370	pluripotency factor	Green

Table 7.1: Transcription factors selected for the overlap analysis using chromatin states. The color code indicated in the last column is used throughout the remainder of this use case.

Human umbilical vein endothelial cell (HUVEC), K652, and NHEK. Figure 7.10 shows the results of the overlap operation, where it can be observed that the *TF* EP300 has the highest amount of overlaps in all BioSources. But an important finding is that the TFs responsible for pluripotency is considerably higher in the *H1-hESC* and *EZH2*, the transcriptional repressor, is almost zero in the same BioSource.

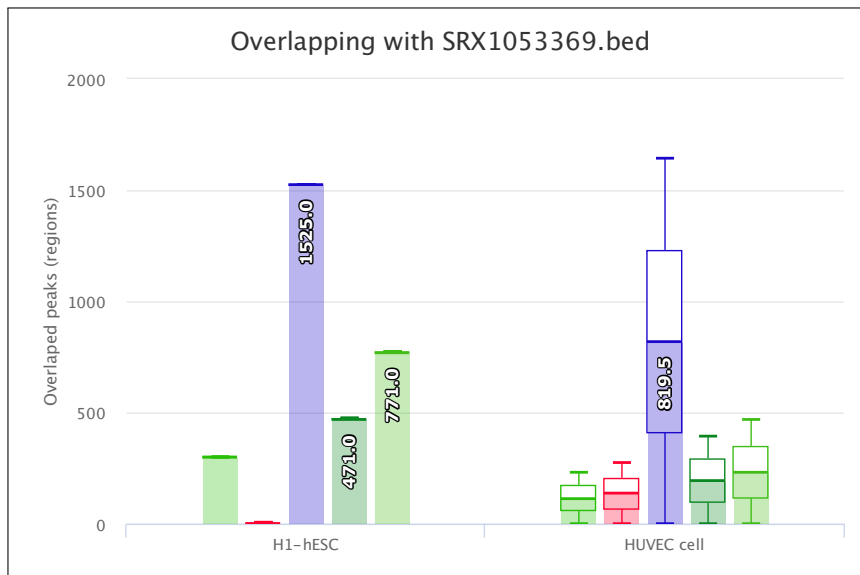


Figure 7.10: Overlapping transcription factors with the chromatin state Strong Enhancer: this chart shows that the TFs responsible for pluripotency have more overlaps in the chromatin state *Strong Enhancer* of the *H1-hESC* than in the *HUVEC*. It also shows a small count of overlaps of the transcriptional repressor *TF* *EZH2* in the *H1-hESC* these chromatin state regions. The author verified that the high variance of some results is due the low quality of some experiments data provided by ChIP-Atlas.

A second analysis is made using the chromatin state *ehn* (from enhancer) that is provided by *ENCODE* project. The result presented in Figure 7.11 allows to draw a similar conclusion. This second analysis contains *embryonic stem cell* and also *induced pluripotent stem cell*, both of which show similar behavior, with a larger number of overlaps in the pluripotency TFs and lowest levels in the transcription repressor. It is interesting to observe that the BioSource *animal ovaries* shows a high count in the repressor *TF* *EZH2*,

which is aberrantly overexpressed in ovarian cancer (Wen *et al.* 2017) and is a target for ovarian cancer (Jones *et al.* 2018).

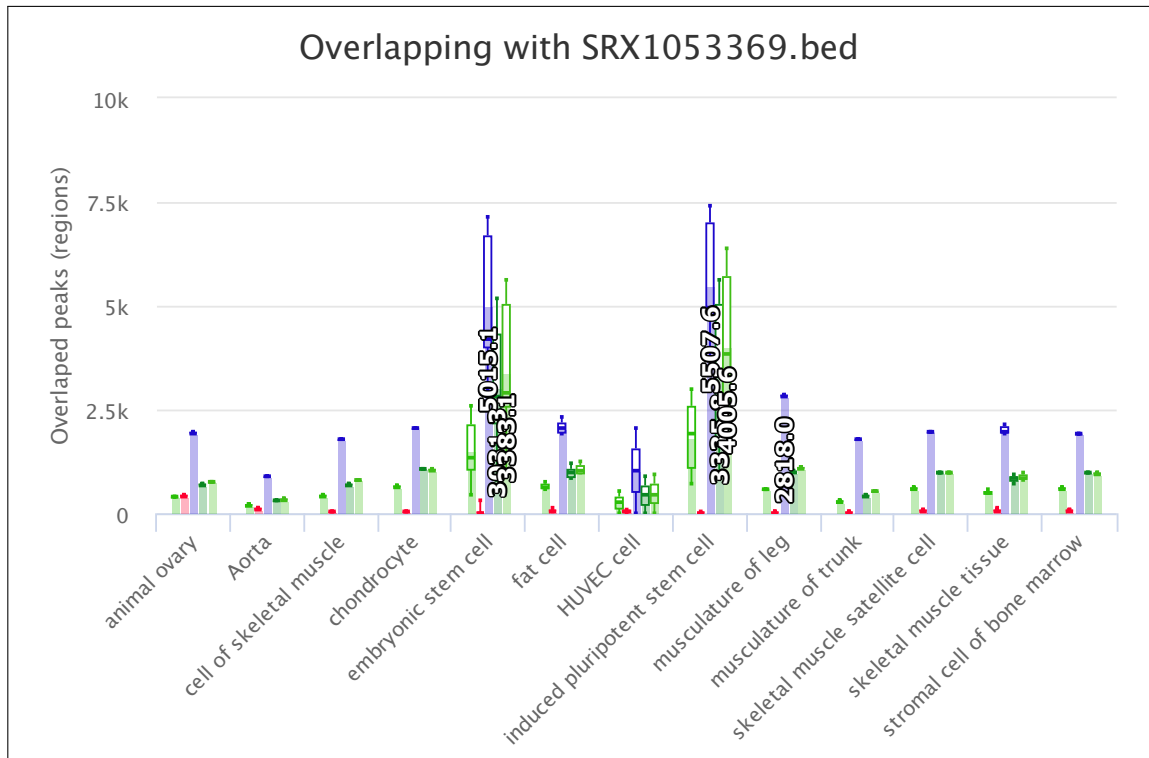


Figure 7.11: Overlapping transcription factors with the chromatin state Enh (Enhancer from ENCODE): this chart shows that the TFs responsible for pluripotency have more overlaps in the chromatin state *Strong Enhancer* of the *embryonic stem cell* and *induced pluripotent stem cell* than in the other BioSources. It also shows a low amount of overlaps of the transcription repressor TF *EZH2* in the two previously mentioned BioSources when considering these chromatin state regions. The author verified that the high variance of some results is due the low quality of some experiments data included in ChIP-Atlas.

While the previous use case re-identified TFs partly responsible for maintaining the pluripotency state in *stem cells*, this use case aimed to visualize the difference of *stem cells* to many other different BioSources using chromatin state information. It should be pointed out that both use cases do not require any programming skills and achieve their goals with only a few mouse clicks in the *DIVE* web interface.

7.5.3 Comparing data from different consortia

This use case describes a framework for comparing datasets from different epigenomic mapping projects. Even that mapping projects follow the standards provided by *IHEC*, Section 5.3.4 explained how batch effects can be introduced in the data by wet lab and data processing procedures. It leads to an important question: *how comparable are the datasets generated by different epigenome mapping projects?*

This use case compares the DNA methylation data from hepatocyte cells from *DEEP* and *CREST*. The comparison of datasets coming from different sources is complicated

by several challenges: (i) the available data is *signal*, not peaks; (ii) due to the data resolution, some files have more than 6 million data points, which make it prohibitive to analyze these data in *DIVE*. For solving these problems, the data is firstly pre-processed using the *DeepBlue* API and then loaded into *DIVE* for visual analysis using CSS data.

Due to these challenges, this use case is composed of two parts: data pre-processing and a visual analysis using *DIVE*. The pre-processing uses a *Python* script for discretizing and filtering the experiments' data. Listing A.15 available in Appendix A.3.3 provides the full source code and complete explanation. In general, this code selects two experiments from each project (*DEEP* and *CREST*), summarize the data in the promoter regions, and for each experiment, and for each experiment file, it constructs two region-sets: one containing hypomethylated regions and the other containing hypermethylated regions. In this use case, hypomethylated are regions which their DNA methylation level is lower or equal to 15 (from a range of 0 to 100), and hypermethylated regions have this value equal or higher to 85. In the end of its execution, the script provides eight *query IDs* that must be loaded in *DIVE*.

The script output is⁶:

```
41_Hf03_LiHe_Ct_WGBS_S_1.THBv2.20150424.GRCh37.cpg.filtered.CG.bedgraph q7702348 q7702349
41_Hf01_LiHe_Ct_WGBS_S_1.THBv2.20150424.GRCh37.cpg.filtered.CG.bedgraph q7702350 q7702351
HPC8_cpg.rate.bedgraph q7702352 q7702353
HPC28_cpg.rate.bedgraph q7702354 q7702355
```

The script output contains the name of the four files, two from *DEEP* and two from *CREST* and the respective *query IDs*. The first *query ID* after the experiment name refers to the promoter regions having a DNA methylation level lower than 15 and the second region references the promoter regions which the DNA methylation level is higher than 85.

The second step of this use case is to load these *query IDs* into *DIVE* for analyzing the data. *DIVE* allows easy and seamless import from the *DeepBlue* *query IDs*. For this purpose, as shown in Figure 7.12, in the *select query* step in *DIVE*, it is possible to load a *DeepBlue* *query ID*. If the user provides the *query ID*, and presses *Display Data* and *DIVE* shows the workflow referenced by this *query ID*. After entering the *query ID* to be loaded, the user has to click on the button *Load Query* to load this query into *DIVE* and use it.

In the previous step, the first of the eight *query IDs* needed for this analysis is loaded. The others are loaded in the *select comparison region-set* step, using the same interface for selecting data from *DeepBlue's* *query ID*.

Due to the large number of *comparison* region-sets, it is advisable to select distinct colors to facilitate interpretation of the generated charts. For changing the colors, click on the region-sets names and in the panel with the information, click on the box on the side of the experiment name for choosing its color. In this use case, the following color coding is used: purple for *CREST* low methylated, green for *DEEP* low methylated, red for *CREST* high methylated, and yellow for *DEEP* high methylated.

Figure 7.13 shows the comparison of these region-sets to regions annotated as *Tss-Biv* (TSS bivalent regions) (Yen and Kellis 2015) that are regions that may act as repressing or activating epigenetic regulators. Bivalent chromatin domains are commonly associated with promoters of genes coding for transcription factors expressed at low levels (Bernstein *et al.* 2006). The main plot shows a strong correlation between the *CREST*

⁶ (the query IDs may vary due to changes in the *DeepBlue* Server)

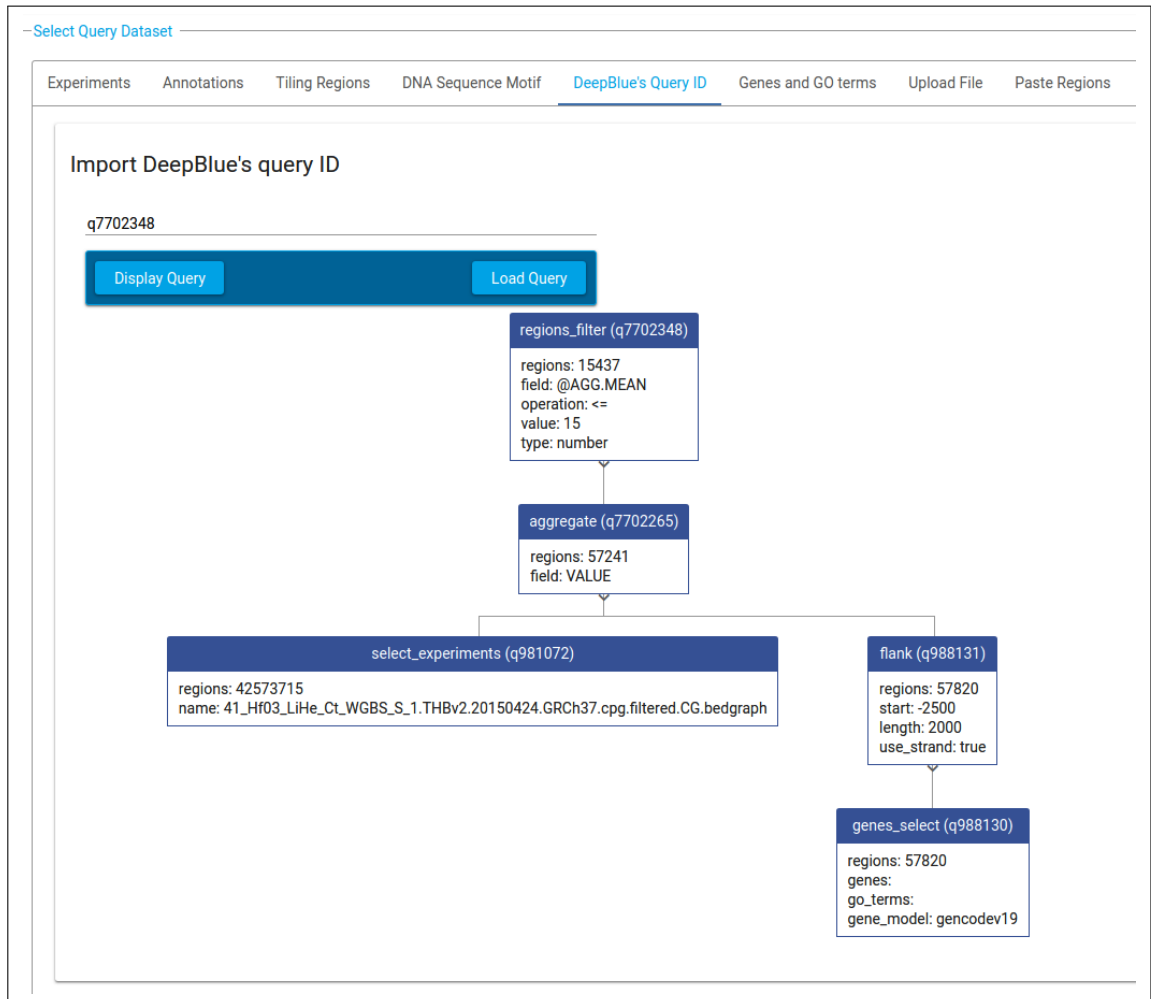


Figure 7.12: Loading a *DeepBlue* query ID into *DIVE*: *DIVE* provides an interface for loading *DeepBlue* queries, showing the workflow referenced by them before loading it.

and *DEEP* experiments in all three analyzed BioSources: blood, brain, and liver. It is possible to observe that both lowly methylated promoter region-sets (blue and green bars) have a larger overlap count and the highly methylated region-sets (red and yellow) have a lower count. Showing that *DEEP* and *CREST* data have a similar pattern.

Figure 7.14 shows the comparison of the region-sets to the chromatin state *enh*, which represents enhancer regions. In this case, the difference of the overlap count between the liver samples and the other BioSource samples is smaller. But more importantly, it is possible to observe the correlation between *DEEP* in this figure. Both projects have a similar overlap count in lowly and highly methylated region-sets for all observed comparisons to the BioSources. This figure also shows another *DIVE* feature: exporting the charts as images.

This use case demonstrates the power of connecting the *DeepBlue* Server API with *DIVE*: users can perform data pre-processing, e.g., converting signal data to disjoint regions using the *DeepBlue* Server API, and then, visually analyse these regions in *DIVE*. It should be pointed out that the *DeepBlue* Server performed more than 1000 overlap

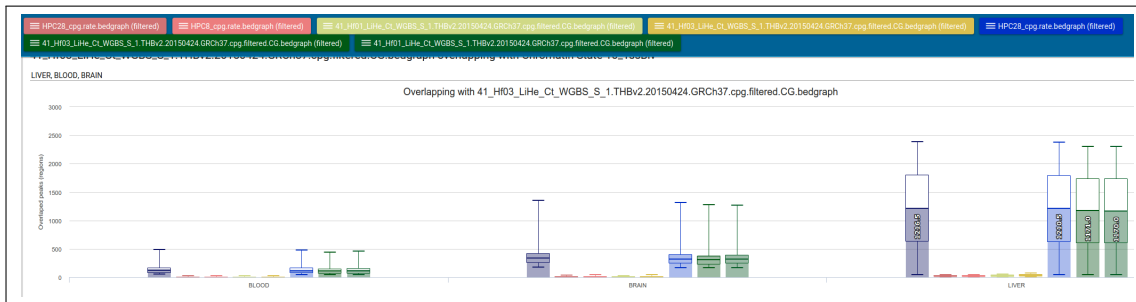


Figure 7.13: Comparing CREST and *DEEP* data: overlapping to TSS bivalent regions: this chart shows the distinct two region-set groups (low and high DNA methylated regions). It is possible to observe that the low methylated regions have a higher number of overlaps in the live samples. In addition, the selected data from *DEEP* correlates to the selected CREST data.

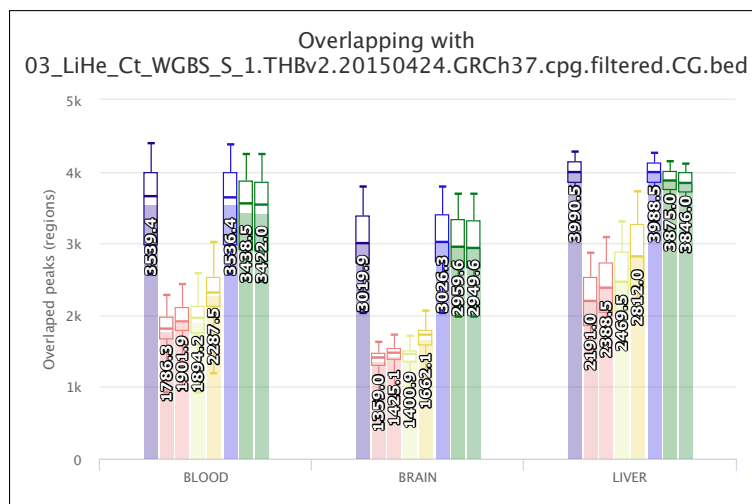


Figure 7.14: Comparing CREST and *DEEP* data: overlapping to enhancer regions: again, it is possible to observe a distinct difference between the low and high methylated regions and the correlation between the CREST and *DEEP* region-sets.

operations to generate the data for the previous chart, where each of the eight generated region-sets were compared to approximately 125 region-sets.

From the data analysis perspective, this use case shows that the analyzed CREST and *DEEP* data correlates when overlapping with the selected BioSources. But it shows a small batch effect, where the CREST samples have very similar results among them and the *DEEP* samples likewise, resulting in a slight difference between these two region-sets groups. In conclusion, these data is comparable for high-level analysis, such as, overlap counts, but higher attention is required for complex data analyses.

7.6 Conclusion

DIVE is a web application for epigenomic data analysis that allows researchers to analyze datasets by comparing them to the thousands of experiments and annotations available in the *DeepBlue Server*.

The region-sets overlapping in *DIVE* is inspired by *EpiExplorer*, but it brings new functionalities, such as guided data analysis, run-time data processing, fully flexible queries, dynamic region-sets, multiple samples per *BioSources*, and multiple region-sets sources. *DIVE* also brings data enrichment: one available enrichment method is the implemented *LOLA* method in the *DeepBlue Server*. But *DIVE* also uses a faster method that is used for guiding users through the data analysis.

DIVE has been used for analyzing hundreds of region-sets. Different use cases (Section 7.5), demonstrate its flexibility, and capabilities for analyzing large-scale epigenomic data in different scenarios.

DIVE is open source, and its modular source code (Appendix A.2.6) is based on components that can be reused to easily extend *DIVE*'s functionalities. Hence, *DIVE* is also a framework for developing tools for large-scale epigenomic data analysis.

Finally, *DIVE* is a unique tool, which merges the tremendous amount of data provided by *DeepBlue*, with its powerful API, in a comprehensive, dynamic, programming-free, and visual web interface for analyzing large-scale epigenomic data.

8

Summary and outlook

IHEC member projects have generated more than 50,000 epigenomic datasets. Analyzing all this data is difficult due to limitations in the current methods for searching, manipulating, and retrieving epigenomic data. The primary objective of the *DeepBlue Epigenomic Data Server* ecosystem is to empower researchers by offering easy-to-use tools for the large-scale analysis of epigenomic data.

The *DeepBlue Server* has more than 160 registered users¹, which were responsible for 5% of all 3,570,000 workflow execution processes, where anonymous users executed the remaining. The *DeepBlue* community is growing, with more than 15 new users in 2019. During 2019, a year which its development was not highly active, more than 120,000 workflow executions were performed (8,000 by registered users and 112,000 by anonymous users).

The *DeepBlue Epigenomic Data Server* ecosystem consists of the following components:

DeepBlue Server	main components, responsible for storing and handling the (epi)genomic data and its metadata, and for processing user requests (Section 4.2).
DeepBlue Populator	imports the metadata entities content and (epi)genomic data to the <i>DeepBlue Server</i> (Section 4.4).
DeepBlueR	R package that streamlines communication between the <i>DeepBlue Server</i> and the <i>R/Bioconductor</i> environment (Chapter 5).
DeepBlue Web Portal	web portal for searching and accessing the data provided by the <i>DeepBlue Server</i> (Chapter 6).
DIVE	web tool for analyzing epigenomic data (Chapter 7).
DeepBlue Middleware	intermediate component realizing the data exchange between <i>DeepBlue Web Portal/DIVE</i> and the <i>DeepBlue Server</i> (Section A.2.5).

Each of the listed components has a defined functionality that covers loading the data into the server, answering user requests, connecting the data with existing tools and libraries, and providing intuitive interfaces for data searching, download, and analysis.

The goal of such a complete ecosystem is to empower researchers with various backgrounds to perform a wide range of analysis tasks. Specifically, the *DeepBlue Epigenomic Data Server* ecosystem targets the following user groups:

Software developers can use the *DeepBlue Server* API for developing or extending their own epigenomic data analysis tools. Examples include, *deepTools* (Ramírez *et al.* 2016) and the BLUEPRINT Data Analysis Portal (J. M. Fernández *et al.* 2016). In addition,

¹ The usage information presented here was obtained on June 20th, 2019

DIVE's code base is strictly organized by components, which facilitates developers to easily extend its functionalities or use its components in other projects.

Experienced data analysts can use the *DeepBlue Server* API directly or through the *DeepBlueR* package for performing complex (epi)genomic data analysis. They can perform data aggregation or enrichment directly on the server and connect these results to the existing data analysis tools using programming languages such as *Python* and *R*.

Biomedical researchers can use *DIVE* for analyzing their epigenomic data in an intuitive and programming-free environment. They can upload their data and compare it to thousands of (epi)genomic experiments and annotations.

The conception and development of the *DeepBlue Epigenomic Data Server* ecosystem along with multiple completed (epi)genomic data analysis brought a better understanding about the needs and requirements in large-scale epigenomic data analysis that is used here for answering the three goals stated in Section 1.1:

- *Organize the public epigenomic data such that researchers can answer their questions more easily*

The proposed solution is to organize the public data with a minimum of mandatory metadata, complemented by optional values that contain information provided by specific epigenome mapping projects. Both mandatory and optional metadata must be indexed, for direct and full-text access, and available through simple operations or by an intuitive web interface.

- *Provide means for finding, operating, and analyzing the existing and currently generated epigenomic data in such a way that matches the current pace of epigenomic data generation*

From the user viewpoint, this objective has been realized by implementing an API that can access multiple files and supports selecting regions of interest from these files in a straightforward set of operations. From the infrastructure perspective, the system must be able to start new instances on demand and the data must be deposited in a scalable database.

- *Empower researchers with methods and techniques for performing large-scale epigenomic data analysis*

This work provides tools for different user profiles and environments. For example, *R/Bioconductor* for users with programming skills and web interfaces for biomedical researchers that need a non-programmatic way of accessing and analyzing the data. Furthermore, these tools cope with the large-data volume, focusing on the global data analysis rather than specific genomic locations.

Coping with the open challenges in large-scale epigenomic data analysis

The *DeepBlue Epigenomic Data Server* and its ecosystem were developed to cope with the existing challenges in the large-scale epigenomic data analysis presented in Chapter 3.5, each of these challenges is addressed in the following paragraphs.

The *DeepBlue Server* provides a unified searching mechanism for the data and metadata generated by the IHEC member projects', and integrates the epigenomic data and metadata of such projects. It allows users to search for the epigenomic data in a single location using standardized operations and metadata content. Furthermore, a web portal is provided facilitating searching and downloading such data.

The *DeepBlue Server* also provides methods for filtering and transforming the data before being downloaded. Users can access this server directly by its API or through its *R/Bioconductor* package in their personal computer for investigating the epigenomic data in environments with limited resources.

The *DeepBlue Server* has been developed to be scalable and cloud-ready. It can handle and provide (epi)genomic data in configurable formats and it handles heterogeneous different data allowing users to specify which columns to be included in the output, as well, to inform the file format when uploading it. Its API allows to access, manipulate, and analyze data from regions of the whole genome, not being fixed to gene or other pre-specified locations. Users define which regions are interesting for their search.

DIVE is a tool for analyzing epigenomic data in a programming-free environment with a user-friendly interface. In this way, the *DeepBlue Epigenomic Data Server* ecosystem copes with the amount of generated epigenomic data, not only in finding and manipulating but also for visualizing and analyzing it.

Existing workflows for epigenomic data processing and analysis are inflexible. Researchers are usually interested in only a specific portion of the epigenomic data for their research. Complex scripts implement these workflows, lacking flexibility for adapting them for the new epigenomic questions. In the current epigenomic studies, new research questions are being addressed, thus, it is not possible to depend on a system with static and pre-processed data. *DeepBlue Server* changes this landscape by providing a flexible API integrated into a vast collection of (epi)genomic data and metadata. With this flexible API, users can learn to access and build workflows for searching, manipulating, enriching, and retrieving terabytes of epigenomic data. For researchers that do not have programming knowledge, the *DeepBlue Epigenomic Data Server* ecosystem offers *DIVE*, which, besides providing a data analysis portal, can also be used for building data filtering workflows in a graphical user interface environment.

With all the characteristics and features presented previously, the *DeepBlue Epigenomic Data Server* ecosystem is setting the state of the art for large-scale epigenomic data analysis.

8.1 Outlook

To stabilize the *DeepBlue Ecosystem* as a long term resource, it was installed in the *German Network for Bioinformatics Infrastructure (de.NBI)* infrastructure. *DeepBlue* is offered to IHEC and it is under scrutiny by this consortia.

Single-cell data analysis has the potential of redefining the (epi)genomic data analysis landscape. It is imperative to be prepared for a dramatic increase in the amount of epigenomic data and new types of entities, such as aggregations of hundreds of cells. Large-scale epigenomic data analysis of today will be the routine analysis in the near future, pushing the limits of tools and methods development even further. Consequently, it is even more important to have powerful resources that cope with this data increase,

and for analyzing all of these data, the *DeepBlue Epigenomic Data Server* requires the following improvements:

- The *DeepBlue Server* can distribute workflow requests over several cluster instances. But the processing of the individual workflows happen in a single server instance. Hence, its processing parallelization must be able to distribute the processing of internal requests for coping with the increasing size of the epigenomic data.
- The required (epi)genomic regions are loaded at the beginning of the workflow processing, consuming memory and preventing the analysis of arbitrarily large epigenomic datasets due to memory limitations. A required approach is to use data iterators that provide the data for processing by the *DeepBlue Server* on demand.
- The quality of data provided by the *DeepBlue Server* depends directly on the data imported by the *Populator*. It is desirable to have a manual or programmatic data curation step. Ideally, this step should remove low-quality datasets, as well, annotate the experiments columns, and fix and organize sample information, such as their BioSources names. The usage of ontologies was a significant step towards a better sample annotation, but more manual work is required.
- The GO terms are imported without the hierarchy, but more complex data selection and analysis can benefit from the use of hierarchy similarly to BioSources.
- Converting region-sets from different genome versions is useful for combining data from various projects. *DeepBlueR* uses an external library for providing such functionality locally, but it is desirable to have such a feature embedded in the *DeepBlue Server*.
- Implementing downstream analysis methods in the *DeepBlue Server* such as clustering and also imputation methods for improving the support in single cell epigenomic data.
- The *DeepBlueR* package does not implement the methods for storing user data in the *DeepBlue Server*. It could be interesting to have this feature, such that users can easily save their analysis results and other datasets in private spaces. Allowing users to perform analysis using their data for comparison, and also sharing it with collaborators.
- *DIVE* has been developed with extensibility in mind. However, *DIVE* still lacks functionalities such as loading DNA sequences in both strands efficiently or performing overlap analysis by genomic regions based on DNA motifs. Besides, currently the data comparison process executed by the user is performed by choosing an epigenetic mark and then the BioSources. It is useful to be able to choose the BioSources and then its epigenetic marks for performing the overlap analysis. Thus, facilitating the analysis that requires comparing different histone marks or chromatin states.
- It is desirable to support other data types, such as *Variant Call Format (VCF)* data for handling *Single Nucleotide Polymorphism (SNP)* information. Furthermore, connecting these new data content with changes in the epigenomic data of the same sample, and the consequences of these changes on the gene expression levels, would culminate in supporting the *Genotype-Tissue Expression (GTEx)* data.

- Currently, the *DeepBlue Epigenomic Data Server* is hosted at the Max Planck Institute for Informatics, with future installation planned in the *de.NBI* infrastructure. For easy scalability and data sharing among users, it would be helpful to have it installed in an external cloud computing system, where new *DeepBlue Servers* can be started on demand, and also the infrastructure costs be shared among its users. An approach for facilitating the installation of the *DeepBlue Server* in different locations and the cloud is to provide self-contained containers for virtualization systems such as *Docker*², *Singularity*³, and *Kubernetes*⁴.

These improvements should not be seen as weak points in the *DeepBlue Epigenomic Data Server* and its ecosystem, but as opportunities of providing better support in the field of computational epigenetics, where better and more robust methods and tools are required for the ever-growing epigenomic data deluge.

² <https://www.docker.com>

³ <https://singularity.lbl.gov/>

⁴ <https://kubernetes.io>



List of Abbreviations

3C	<i>Chromosome Conformation Capture</i>
450K	<i>Illumina Infinium HumanMethylation450 BeadChip</i>
4C	<i>Circularized Chromosome Conformation Capture</i>
5C	<i>Carbon-Copy Chromosome Conformation Capture</i>
ATAC-seq	<i>Assay for Transposase-Accessible Chromatin using Sequencing</i>
AUC	<i>Area under Curve</i>
BAM	<i>Binary Alignment/Map</i>
BED	<i>Browser Extensible Data</i>
BMBF	<i>German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung)</i>
bp	<i>basepair</i>
cDNA	<i>complementary DNA</i>
CEEHRC	<i>Canadian Epigenetics Environment and Health Research Consortium</i>
ChIP	<i>Chromatin Immunoprecipitation</i>
ChIP-seq	<i>Chromatin Immunoprecipitation-Sequencing</i>
chr	<i>Chromosome</i>
CL	<i>Cell Type Ontology</i>
CRF	<i>chromatin-remodeling factor</i>
CSS	<i>Chromatin State Segmentation</i>
DDBJ	<i>DNA Data Bank of Japan</i>
de.NBI	<i>German Network for Bioinformatics Infrastructure</i>
DEEP	<i>Deutsches Epigenom Programm</i>
DMR	<i>Differentially Methylated Region</i>
DNaseI-seq	<i>DNaseI-Sequencing</i>
EFO	<i>Experimental Factor Ontology</i>

EGA	European Genome-phenome Archive
ENA	European Nucleotide Archive
ENCODE	The Encyclopedia of DNA Elements
EpiRR	The Epigenome Reference Registry
eRNA	Enhancer RNA
ESC	embryonic stem cell
EWAS	Epigenome-Wide Association Study
FAIRE-seq	formaldehyde-assisted isolation of regulatory elements
FPKM	fragments per kilobase of transcript per million mapped reads
GEO	Gene Expression Omnibus
GO	Gene Ontology
GTEx	Genotype-Tissue Expression
HMM	Hidden Markov Model
HSC	Hematopoietic Stem Cell
HUVEC	Human umbilical vein endothelial cell
ICGC	International Cancer Genome Consortium
IHEC	International Human Epigenome Consortium
iPSC	Induced Pluripotent Stem Cell
JIT	just-in-time
LIMS	Laboratory Information Management System
LMR	Low-Methylated Region
lncRNA	Long Non-Coding RNA
LOLA	Locus Overlap Enrichment Analysis
LRU	least recently used
MeDIP-seq	Methylated DNA Immunoprecipitation Sequencing
miRNA	Micro RNA
MNase-seq	Micrococcal Nuclease Sequencing
MRE	methylation-sensitive restriction enzyme digestion
mRNA	Messenger RNA
μ WGBS	Low-input Whole Genome Bisulfite Sequencing
NCBI	National Center for Biotechnology Information
ncRNA	Non-Coding RNA
NGS	Next Generation Sequencing
NOME-seq	Nucleosome Occupancy and Methylome-Sequencing
nt	Nucleotide
OWL	Web Ontology Language

PCA	<i>Principal Component Analysis</i>
PCR	<i>Polymerase Chain Reaction</i>
QC	<i>Quality Control</i>
REMC	<i>NIH Roadmap Epigenomics Mapping Consortium</i>
RNA-seq	<i>RNA-sequencing</i>
RNAi	<i>RNA Interference</i>
RNAPII	<i>RNA polymerase II</i>
RPC	<i>Remote Procedure Call</i>
RRBS	<i>Reduced Representation Bisulfite Sequencing</i>
scWGBS	<i>Single-Cell Whole Genome Bisulfite Sequencing</i>
siRNA	<i>Small Interfering RNA</i>
snoRNA	<i>Small Nucleolar RNA</i>
SNP	<i>Single Nucleotide Polymorphism</i>
SRA	<i>Sequence Read Archives</i>
SVA	<i>Surrogate Variable Analysis</i>
SVM	<i>Support Vector Machine</i>
TCA	<i>Transcription Coactivator</i>
TCGA	<i>The Cancer Genome Atlas</i>
TE	<i>transposable Element</i>
TF	<i>transcription factor</i>
TFBS	<i>Transcription Factor Binding Site</i>
TPM	<i>transcript per million</i>
TSS	<i>Transcription Start Site</i>
UBERON	<i>Uber Anatomy Ontology</i>
UI	<i>User Interface</i>
UMR	<i>Unmethylated Region</i>
URL	<i>Uniform Resource Locator</i>
VCF	<i>Variant Call Format</i>
WGBS	<i>Whole Genome Bisulfite Sequencing</i>
WGSBS	<i>Whole genome shotgun bisulfite sequencing</i>
WIG	<i>Wiggle Track Format</i>

Glossary

API

An Application Programming Interface “provides an abstraction for a problem and specifies how clients should interact with the software components that implement a solution to that problem” (Reddy 2011).

BED

BED (Browser Extensible Data) is a data format that provides a flexible way to define the (epi)genomic data. BED files are based on columns, having three required fields (chromosome, start, and end) and additional optional fields. The number of fields per line must be consistent across the entire file.

bedGraph

The bedGraph format allows display of continuous-valued data in track format. This track type is similar to the WIG format, but unlike the WIG format, data exported in the bedGraph format are preserved in their original state.

bigBed

A BigBed file is created initially from BED type files. The resulting bigBed files are in an indexed binary format.

biomarker

A biomarker is a measurable (molecular) indicator for a given (disease) condition or state.

BioSource

A BioSource defines the biological source of a sample. The biological source can be a cell line, a cell type, tissue, or organ.

bitmap

A bitmap is a vector of bits, where the values are in the form of *true* or *false*. One of the main advantages of bitmaps is lower memory and disk space consumed and fast computational operations in its content.

BSON

BSON is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays (<http://bsonspec.org/>). In BSON, the documents are binary-encoded, improving its used memory and disk space, as well, the access to the data content.

CpG

The CpG is a DNA dinucleotide consisting of cytosine followed by guanine, linked by phosphate.

CpG island

A CpG island is a genomic region particularly rich in CpG dinucleotides. Multiple alternative definitions and algorithms for defining them exist. The most widely used definition was introduced by Gardiner-Garden and Frommer 1987.

dictionary

A dictionary, or associative array, map, hash table, is an abstract data type composed of a collection of key and value pairs, such that each possible key appears at most once in the collection.

DNaseI hypersensitive site

DNaseI hypersensitive site is an accessible genomic region frequently cleaved by DNaseI nucleases. Typically indicates open chromatin and low nucleosome occupancy.

enhancer

Enhancer is a genomic region containing TFBSs located distal or proximal to gene promoters involved in transcriptional activation. Active enhancers are typically characterized by the presence of H3K4me1, H3K4me2, H3K27ac, DNaseI hypersensitivity and p300 occupancy (Ong and Corces 2011).

epigenetic mark

An epigenetic mark is a chromatin modifications such as DNA methylation and histone post-translation modifications that influentiate the cell's gene expression and its phenotype.

FASTA

FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which nucleotides or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences.

gene-model

A gene-model is a set of genes imported from a *GENCODE Harrow et al. 2012; Derrien et al. 2012* release, for example, GENCODE v19.

GTF

GTF/GFF (General Feature Format) format consists of one line per feature, each containing 9 columns of data, plus optional track definition lines (<https://www.ensembl.org/info/website/upload/gff.html>).

haematopoiesis

Haematopoiesis is the Process by which all mature blood cells are produced.

hypermethylated

A hypermethylated region is region which contains a higher DNA methylation level in a given condition compared to a reference condition.

hypomethylated

A hypomethylated region is region which contains a lower DNA methylation level in a given condition compared to a reference condition.

index

An index is a data structure that organizes the data that can be searchable efficiently.

JSON

JavaScript Object Notation or JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

Lua

Lua is a powerful, dynamic and light-weight programming language. It may be embedded or used as a general-purpose, stand-alone language.

LuaJIT

LuaJIT is a Just-In-Time Compiler (JIT) for the Lua programming language. It is an open-source project and it is available at <http://luajit.org/>.

LZO

Lempel–Ziv–Oberhumer (LZO) is a lossless data compression algorithm that is focused on decompression speed developed (<http://www.oberhumer.com/opensource/lzo>).

meta-field

Meta-fields are pseudo data columns that do not provide access to the (epi)genomic data column, but to its metadata content, such experiment name or biosource. Using them, it is also possible to execute a command in the context of the individual genomic region, rather than in the set of the regions. The Table 4.5.5 shows all meta-fields present in the *DeepBlue Server*.

peak

A peak is a discrete region in the genome, defined by the chromosome, start, and end. Peaks are used to annotate genome regions, for example, CpG Islands, histone modification locations, open chromatin, transcription factors binding site, and genes.

populator

The populator is a tool for including metadata and data into DeepBlue.

promoter

A promoter is a genomic region located near the TSS to which the transcriptional machinery is recruited. Typically defined to extend from a few kilobases upstream of the TSS to a few hundred bases downstream of the TSS.

region

Region is the central data structure for organizing the (epi)genomic data in the DeepBlue Server. It contains the region start, end, and additional optional fields. Regions are organized in lists, where each list contains the regions of a given chromosome.

regular expression

A regular expression or regex is a formal language that define a search pattern. Usually this pattern is then used by string searching algorithms.

signal

Signal is a set of continuously genomic locations that are used for annotating the genome. Signal data is used with histone modifications, open chromatin, and Transcription factors binding sites.

user_key

A user_key is private key that is used by a user for accessing DeepBlue.

WIG

The bigWig is a data format useful for dense and continuous data. BigWig files are created from WIG type files. The bigWig files are in an indexed binary format.

WIG

The wiggle (WIG) format is a data format for storing and displaying dense, continuous, and equally sized (epi)genomic data. Wiggle data must be continuous and consist of equally sized elements. If your data is sparse or contains elements of varying sizes, use the bedGraph format instead of the wiggle format.

XML-RPC

XML-RPC is a Remote Procedure Call method that uses XML passed via HTTP as a transport. Its specification is available at <http://xmlrpc.scripting.com/spec.html>.



A.1 Data imported by the DeepBlue Server

This section gives an overview of the data available in the *DeepBlue Server*. For more specific questions, it is suggested to access the *DeepBlue Web Portal*(<http://deepblue.mpi-inf.mpg.de>).

Table A.1 contains the genome assemblies available in the *DeepBlue Server*.

Genome	URL
<i>hg19</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=hg19
<i>mm9</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=mm9
<i>mm10</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=mm10
<i>hs37d5</i>	http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/
<i>GRCm38</i>	http://www.ensembl.org/Mus_musculus/Info/Index

Table A.1: Genome assemblies available in the DeepBlue Server.

Table A.2 contains the gene models available in the *DeepBlue Server*.

Gene Model	URL
<i>hg19</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=hg19
<i>mm9</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=mm9
<i>mm10</i>	http://genome.ucsc.edu/cgi-bin/hgGateway?db=mm10
<i>hs37d5</i>	http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/
<i>GRCm38</i>	http://www.ensembl.org/Mus_musculus/Info/Index

Table A.2: Gene models available in the DeepBlue Server.

Table A.3 presents the number of experiment files imported into the *DeepBlue Server*. With the exception of *DEEP* (Private), this data is available for all users, totaling 58402 publicly available experiment files.

Project	Number of files	Project URL
<i>DEEP</i> (Private)	4073	www.deutsches-epigenom-programm.de/
<i>DEEP</i> (from IHEC data portal)	341	www.deutsches-epigenom-programm.de/
BLUEPRINT Epigenome	5975	www.blueprint-epigenome.eu
ENCODE	28484	www.encodeproject.orgXXX
REMC	5633	www.roadmapepigenomics.org
CREST	299	crest-ihec.jp
CEEHRC	2696	www.epigenomes.ca
ChIP-Atlas	14974	chip-atlas.org

Table A.3: Experiment files available in the DeepBlue Server.

Table A.4 presents the annotations available in the *DeepBlue Server* with their respective genome. Gene and promoter annotations can be obtained from the gene-models, but their files are kept in the annotations for backward compatibility with scripts that were developed before this feature was implemented.

Annotation	Genome
repeat_masker	hg19
Cpg Islands	hg19
Genes	hg19
Promoters	hg19
Probes450k	hg19
lamin_b1	hg19
conservation_primates	hg19
conservation_placental	hg19
RepeatFree_1kb_autosomes	mm10
RepeatMasked_1kb_autosomes	mm10
gene_protocod_3kbprom_autosomes	mm10
gene_protocod_full_autosomes	mm10
transcripts_protocod_3kbprom_autosomes	mm10
transcripts_lincRNA_full_autosomes	mm10
transcripts_miRNA_full_autosomes	mm10
transcripts_snRNA_full_autosomes	mm10
transcripts_snoRNA_full_autosomes	mm10
Blueprint Ensembl Regulatory Build	GRCh38
CpG Islands	GRCh38
Promoters	GRCh38

Table A.4: Experiment files available in the DeepBlue Server.

Table A.5 presents the imported ontologies used for naming the BioSources.

Ontology	Description	URL	Date/Version
CL	Controlled vocabulary for cell types in animals	http://www.ontobee.org/ontology/CL	2016-02-01
EFO	Systematic description of many experimental variables available in EBI databases	https://www.ebi.ac.uk/efo/	version 2.72
UBERON	Integrated cross-species anatomy ontology covering animals and bridging multiple species-specific ontologies	http://uberon.github.io/	2016-05-11

Table A.5: Experiment files available in the DeepBlue Server.

Table A.6, Table A.7, and Table A.8 provide the list of chromatin states provided by REMC, BLUEPRINT, and ENCODE CSS files.

The histone mark names are imported from *H1stome database* (Khare *et al.* 2011) and other epigenomic targets are imported from the targets page of the ENCODE Search (<https://www.encodeproject.org/targets/>).

State No.	Mnemonic	Description
1	TssA	Active TSS
2	TssAFlnk	Flanking Active TSS
3	TxFlnk	Transcr. at gene 5' and 3'
4	Tx	Strong transcription
5	TxWk	Weak transcription
6	EnhG	Genic enhancers
7	Enh	Enhancers
8	ZNF/Rpts	ZNF genes & repeats
9	Het	Heterochromatin
10	TssBiv	Bivalent/Poised TSS
11	BivFlnk	Flanking Bivalent TSS/Enh
12	EnhBiv	Bivalent Enhancer
13	ReprPC	Repressed PolyComb
14	ReprPCWk	Weak Repressed PolyComb
15	Quies	Quiescent/Low

Table A.6: Chromatin States used in the ROADMAP project for the genome *hg19*. Source: https://egg2.wustl.edu/roadmap/web_portal/chr_state_learning.html.

State No.	Description
1	Repressed Polycomb High signal <i>H3K27me3</i>
2	Repressed Polycomb Low signal <i>H3K27me3</i>
3	Low signal
4	Heterochromatin High Signal <i>H3K9me3</i>
5	Transcription High signal <i>H3K36me3</i>
6	Transcription Low signal <i>H3K36me3</i>
7	Genic Enhancer High Signal <i>H3K4me1</i> & <i>H3K36me3</i>
8	Enhancer High Signal <i>H3K4me1</i>
9	Active Enhancer High Signal <i>H3K4me1</i> & <i>H3K27Ac</i>
10	Distal Active Promoter (2Kb) High Signal <i>H3K4me3</i> & <i>H3K27Ac</i> & e1
11	Active TSS High Signal <i>H3K4me3</i> & <i>H3K4me1</i>
12	Active TSS High Signal <i>H3K4me3</i> & <i>H3K27Ac</i>

Table A.7: Chromatin States used in the BLUEPRINT project for the genome *GRCh38*. Source: http://ftp.ebi.ac.uk/pub/databases/blueprint/releases/20140811/homo_sapiens/secondary_analysis/Segmentation_of_ChIP-Seq_data/README_ChromHmm_release_20140811.

A.2 Implementation details

This section provides insights on the *DeepBlue Server* API implementation. The entire *DeepBlue Epigenomic Data Server* ecosystem source is open and free to use, under the GPL 3 (GNU General Public License version 3) restrictions. The following sections presents the main parts, methods, and source code of the *DeepBlue Epigenomic Data Server* ecosystem, where the source code is available in the following repositories:

Server <https://github.com/MPIIComputationalEpigenetics/DeepBlue>
 Populator <https://github.com/MPIIComputationalEpigenetics/DeepBlue-Populator>
 Middleware <https://github.com/MPIIComputationalEpigenetics/DeepBlue-Middleware>
 Web Dashboard <https://github.com/MPIIComputationalEpigenetics/DeepBlue-Web>
 Dive <https://github.com/MPIIComputationalEpigenetics/Dive>

State No.	Description
1	Active Promoter
2	Weak Promoter
3	Inactive/poised Promoter
4	Strong enhancer
5	Strong enhancer
6	Weak/poised enhancer
7	Weak/poised enhancer
8	Insulator
9	Transcriptional transition
10	Transcriptional elongation
11	Weak transcribed
12	Polycomb-repressed
13	Heterochromatin; low signal
14	Repetitive/Copy Number Variation
15	Repetitive/Copy Number Variation

Table A.8: Chromatin States used in the ENCODE project for the genome *hg19*. Source: <http://rohsdb.cmb.usc.edu/GBshape/cgi-bin/hgTrackUi?db=hg19&g=wgEncodeBroadHmm>.

R Package <http://bioconductor.org/packages/release/bioc/html/DeepBlueR.html>

A.2.1 DeepBlue Server

The *DeepBlue Server* is implemented in C/C++11, having around 51,000 lines of code. To ensure its quality, the *DeepBlue Server* has 251 integration tests implemented in *Python*. Each integration tests performs a set of operations in the *DeepBlue Server* and its result is compared to the expected output. The *DeepBlue Server* uses the following libraries:

- **MonbgoDB cxx driver legacy** (1.1.2) for accessing the MongoDB server - github.com/mongodb/mongo-cxx-driver/archive/legacy-1.1.2.zip
- **Boost C++ libraries** provides miscellaneous support, such as the network connection and data interface compression - www.boost.org
- **Expat XML Parser** for reading the XML content from the XML-RPC requests - libexpat.github.io
- **Bzip2** for high compression rates used to store fast enrichment bitmap regions - sourceforge.net/projects/bzip2/
- **minilzo** for real-time compressing and loading regions blocks from the database - www.oberhumer.com/opensource/lzo/
- **cppformat** for formatting strings - github.com/cppformat/cppformat
- **LuaJIT** provides the Lua embedded environment - luajit.org/
- **StrTk** for tokenizing string, used in the files readers - www.partow.net/programming/strtk/
- **jemalloc** for memory allocation - github.com/jemalloc/jemalloc
- **urdl** for accessing external URLs - think-async.com/Urdl/doc/html/index.html
- **MDBQ** is highly adapted for the *DeepBlue Server* needs - github.com/temporaer/MDBQ

The *DeepBlue Server* uses the *MongoDB* version 3.2. It is recommended to not use version 3.4 or later, because they are not compatible with the *MongoDB C++ Driver* (Legacy version 1.1.2) that it is used.

Data layers

For organizing the (epi)genomic data collection, the *DeepBlue Server* organizes its entire data content, not only (epi)genomic data, into four layers:

Relationships	contains the controlled vocabularies with the terms used in the metadata.
Metadata	describes the data.
Data	contains the regions and are annotated by the metadata.
Genomic Regions	compose the data. They are the regions that are operated by the operations and returned to users.

This data division has proven very useful and flexible. For example, it easily enables obtaining the genomic locations that contain information from a given relationship, for example, a gene ontology term or the DNA methylation from a biosource. It also allows for building more complex relationships, for example, to map gene expression files to genomic locations using a gene model. Figure A.1 shows the four data layers with their components.

Relationship	BioSource Ontologies	Controlled vocabularies	Genome assembly	Gene Ontology
Metadata	Epigenomic Experiments ChIP-seq, NoME, WGBS, RRBS		Genomic Annotations CpG Islands, Repetitive elements, DNA motifs	Gene Model Gencode 21, Gencode v22
Data	(Epi-)genomic elements Regions of interest with information			Gene Expression Quantification Genes NPTN, NOX4
Location	Genomic Regions chromosome, start, end			

Figure A.1: DeepBlue Server Data Layer: DeepBlue has 4 layers of data: *Relationships*, *Metadata*, *Data*, and *Genomic Regions*. From bottom to up: The Genomic Regions compose the data that is described by the metadata using the relationship items.

Due to this data division, it is possible to combine information from different types of data for answering users' queries. Figure A.2 shows how the *DeepBlue Server* integrate data from different layers to answer the hypothetical question that select all genes that match que following criteria:

- Chromosome 1, from the genomic location 0 to 70,000
- Annotated with a GO term, for example, a biological process that we want to analyze
- Overlap with some ChIP-seq peaks, like a *TFBS*
- Highly methylated regions, with an average DNA ,methylation level above 0.80

The data integration and operations are possible because ultimately all data are translated to regions, which are the central data structure in the *DeepBlue Server*.

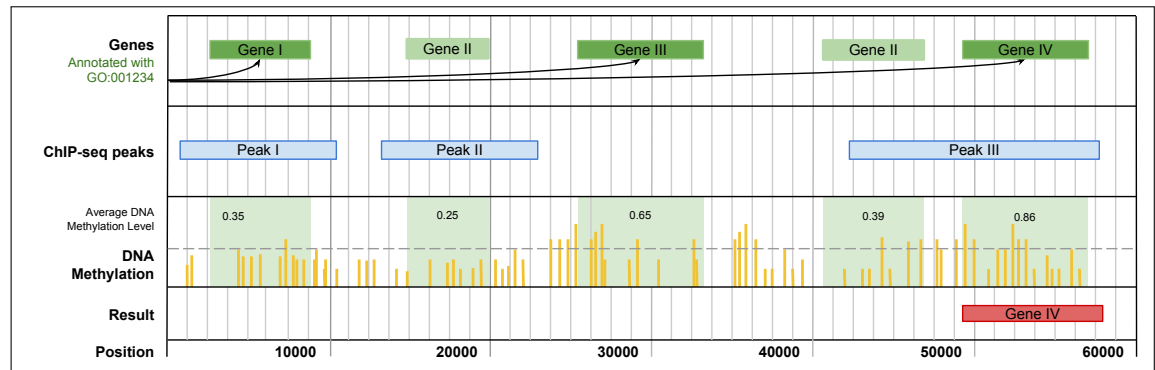


Figure A.2: DeepBlue Server integrates heterogeneous data for answering users' queries, for example, the hypothetical question that select all genes that match the following criteria: (a) Chromosome 1, in the genomic location 0 to 70,000; (b) Annotated with a GO term, for example, a biological process that we want to analyze; (c) overlap with some ChIP-seq peaks, like a Transcription factor binding sites; (d) average DNA methylation level > 0.80 , it means, regions highly methylated. In this case, returning the hypothetical *gene IV*.

Regions

Regions are the most important data structure in the *DeepBlue Server*. They are implemented via the *AbstractRegion* interface and extended for special cases in file *datatype-s/regions.hpp*.

The most frequently used implementation classes are *SimpleRegion* for regions containing only the dataset ID, chromosome, start, and end; *BedRegion* for all BED regions, *WigRegion* for all WIG regions, *GeneRegion* for the genes, and *AggregateRegion* for aggregation results.

It is important to note that *Region* instances are responsible for the most significant part of the memory used by the *DeepBlue Server*, making it imperative that they are optimized for reducing their memory consumption. Hence, their different implementation allows for tuning their structure and methods in the for faster data access and lower memory consumption.

The *DeepBlue Server* loads all regions used by a workflow during the beginning of the workflow execution. Thus, if a workflow loads a set of regions followed by a filtering operation, the *DeepBlue Server* loads all regions from the *MongoDB* and then, the *DeepBlue Server* applies the filter on them. The loading process is executed in parallel, divided by the region genome and chromosome, requiring many small memory allocations, one for each region. For the memory allocation and management tasks, the *DeepBlue Server* uses the library *jemalloc* that coordinates the memory allocation, improving the regions loading time as well as decreasing the memory fragmentation and consequently, the total memory use.

An essential future improvement in the *DeepBlue Server* lies in replacing the naïve approach of loading all the regions at the beginning of the workflow execution by an iterator that loads these regions on demand. It is possible but requires notable changes

in the *DeepBlue Server* implementation, demanding significant software engineering efforts.

DeepBlue operation execution

This section describes how the *DeepBlue Server* receives an operation execution request from the user, how it is processed, and returned, with the central classes involved in the process.

When the *DeepBlue Server* is executed, it starts (*main.cpp*) an *HTTP* server (*httpd/server.cpp*) that listens for users requests on the configured port (defined by the command line parameter *-P/-port*). When a user sends a request, the server instantiates a *Connection* object (*httpd/connection.hpp*) that is responsible for reading the data from the socket, reading the *HTTP* headers (*httpd/request_parser.cpp*), and parsing the request parameters that are encapsulated in an *XML* (*httpd/xmlrpc_parser.cpp*) content.

The *DeepBlue Server XML* parser instantiates an *XmlrpcRequest* (*httpd/xmlrpc_request.hpp*) object. This object contains the operation request parameters: operation name and its parameters. The *XmlrpcRequestHandler* (*httpd/xmlrpc_request.cpp*) deliver *XmlrpcRequest* object to the *Processing Engine* (*engine/engine.hpp*) that executes the operation.

The *Processing Engine* (*engine/engine.cpp*) obtains the operation name and parameters from the *XmlrpcRequest* object. First, it verifies if an operation with the given name exists. Next, it verifies if the user request contains the correct parameters. An error is returned if any of the verifications fail, otherwise, the *Processing Engine* executes the class which implements the operation.

The *DeepBlue Server* operation implementations are located in the *commands/* directory, where each operation is implemented in its own source code file. All operations inherit the class *Command* (*engine/commands.cpp*). The operation description, input parameters, and return types are defined in the own operation implementation class. The operation logic is implemented in the *run()* method, which receives the operation parameters and an object for storing its results. The *run()* method returns *true* when the operation was successfully executed, otherwise it returns *false* and an error message is inserted in the result object.

Finally, the *XmlrpcRequestHandler* (*httpd/xmlrpc_request.cpp*) verifies the result content, and return a *XML* containing the serialized result. The content is returned to the user by the *Connection* object and the connection is closed.

Start-up parameters

It is possible to obtain all *DeepBlue Server* start-up parameters by executing the software with the parameter *-help* or *-H*. The essential parameters are:

- T/-threads* : how many threads the XML-RPC server has in this *DeepBlue Server* instance. The default value is 10, and the XML-RPC server can be switch off by using the value 0.
- R/-processing_threads* : how many concurrent workflow processing requests are processed simultaneously in the in this *DeepBlue Server* instance. The default value is 4, and no processing thread is started when using the value 0.
- O/-processing_max_memory* : the maximum memory available for processing each workflow request. Its default value is 8589934592 (8 gigabytes). It is important that

its parameter value multiplied by the *processing_threads* parameters do not exceed more than 80% of the total available memory of the computer server. This value is important to prevent the abuse of computational resources and can be changed for individual registered users.

`-l/-old_request_age_in_sec` : how old, in seconds, must be a request result for being removed by the janitor. The default value is 2592000 that corresponds to 30 days. Higher values can be used if data storage is not an issue.

Through these parameters it is easy to configure an instance that only process users requests through the XML-RPC server and store the workflow requests in the processing queue that is processed by other instances configured for only processing workflow requests.

DeepBlue Server resources requirements

The required resources correlate directly with the amount of data stored in *DeepBlue Server*. For example, a basic setup with 7,000 peak files (BED) requires less than 30 gigabytes of disk space, but installations with signal data (WIG) may require terabytes of disk space.

As a rule of thumb, each file containing peak regions in the BED format requires two times its compressed size when stored in the *DeepBlue Server*. For example, a compressed file containing peaks in the BED format has the size of 3 megabytes, requires 6 megabytes to be stored in the *DeepBlue Server*. The overhead is lower for signal files, demanding 1.5 times their size, but it is necessary to remember that these files are two or three orders of magnitude larger, where their sizes span from 100 megabytes to gigabytes.

MongoDB is a memory demanding software. The main *DeepBlue Server* installation stores 15 terabytes of signal files and 500 gigabytes of peak files. This does not include the overhead for storing and organizing the data, e.g., the experiments data objects and indexes. In the total, the current *DeepBlue Server* MongoDB instance uses 27 terabytes of disk space, 280 gigabytes of RAM, including 200 gigabytes for the data indexes. It is important that the indexes fit in the memory, otherwise the system will suffer a big performance impact.

The central *DeepBlue Server* installation is executed with 16 threads for the XML-RPC server, eight threads for workflow processing, allowing each workflow processing to use until 16 gigabytes of RAM, which is enough to hold more than 500 million regions. The server in full usage requires 128 gigabytes of memory. The *DeepBlue Server* also requires approximately 5 gigabytes of memory for caching highly accessed data, such as BioSources names and relationship, experiments metadata, and genome sequences.

Even with higher memory requirements, a *DeepBlue Server* instance can be hosted in a computational server with 32 gigabytes of RAM. It is only required to inform in the startup parameters the number of parallel workflows processing requests and the maximum amount of memory that they can use. Besides, many hardware can be combined in a cluster for increasing the throughput of executed processing requests.

Access methods

The communication to the *DeepBlue Server* is performed through the XML-RPC and *RESTful* protocols. A XML-RPC server is implemented in the *DeepBlue Server* and *RESTful* requests are converted to XML-RPC requests by the Middleware (Section A.2.5). The XML-RPC requests must be made to the URL <http://deepblue.mpi-inf.mpg.de/xmlrpc> and the *RESTful* requests to <http://deepblue.mpi-inf.mpg.de/api>.

Data Formats

The *DeepBlue Server* supports data files in the following formats:

BED/Columns Format is used for peak experiments and annotation files. The content of this file must be described using *Column types* controlled vocabulary terms.

wig is used for experiments signal data

bedgraph is used for experiments signal data

fasta is used for genome sequences

gtf is use for importing gene-models from *GENCODE*

cufflinks is used for gene expression data

grape2 is used for gene expression data

salmon is used for gene expression data

Index overhead in the regions blocks

The experiments regions are organized in blocks for reducing the database index overhead and improve the data transfer between the MongoDB instance and the *DeepBlue Server* (Section 4.3.4). Different block sizes were analyzed during the development of the *DeepBlue Server*. Table A.9 summarizes the overhead values from three cases: not using region blocks, blocks containing 1000 regions, and blocks containing 100 regions.

Blocks Size	Regions Count	Object Count	Index Size	Overhead per Region
Without	13,254,208	13,254,208	1,787 megabytes	~ 141 bytes
1000	13,254,208	14,727	17 megabytes	~ 0.3 bytes
100	13,254,208	133,975	3 megabytes	~ 1.3 bytes

Table A.9: Index overhead in different region blocks size: this table shows the overhead generated by indexing the regions individually, by blocks of 1,000 regions, and 100. Even that blocks of 100 regions shows a higher overhead than blocks of 1,000 regions, this value is used because it shows a good tradeoff between index overhead and data retrieval speed.

Currently, *DeepBlue Server* uses blocks of 100 regions, which presents the best tradeoff between the index overhead and quickness for finding the required experiment regions inside a block. The block size is important because if a block is too large, it may delay the data access when only a fraction of the regions block is needed.

Embedded Lua Interpreter

The *DeepBlue Server* embedded a Lua¹ interpreter that is used by the meta-field (@CALCULATED) and the calculated column type (Section 4.5.5). Lua is a scripting programming language that is easy to learn and use. It is fast, has low memory consumption footprint, and it is easy to be embedded in other software, such as in the *DeepBlue Server*.

Rather than using the original Lua interpreter, the *DeepBlue Server* uses LuaJIT (<http://luajit.org/>), which implements Lua version 5.1 functionalities. The main reason for using LuaJIT is because it contains a *just-in-time* (JIT) compiler that compiles the Lua code to machine code at run-time, allowing faster execution of the Lua code. In the *DeepBlue Server*, the Lua environment(*lua/sandbox.cpp*) is executed in a sandbox. Due to security reasons, the sandbox limits the access to the original Lua API not allowing users to access all its commands.

Table A.10 list the most useful commands available in the *DeepBlue's* Lua environment. A complete Lua reference can be found at <https://www.lua.org/manual/5.1/manual.html>.

A.2.2 DeepBlue Populator

The *Populator* is developed in *Python 2*. It uses the MongoDB database for storing the files metadata that should be imported into the *DeepBlue Server* and for tracking which experiments were already imported. For executing the *Populator*, it is required to install the libraries *PyMongo* (interface between *Populator* and the MongoDB database), *requests* (facilitators for downloading files), and *flatdict* (useful data structures).

The *Populator* must configured using its configuration file (*src/settings.py*):

- variable *ROOT* contains the full path to the populator source code
- variables *MDB_HOST* and *MDB_PORT* contain the URL and port of the used MongoDB instance
- variables *DEEPBLUE_HOST* and *DEEPBLUE_PORT* contain the URL and port of the target *DeepBlue Server* instance
- variables *max_downloads* and *max_threads* define how many experiment downloads and insertions can be executed simultaneously.

The *Populator* entry point is the *src/main.py* file. Just execute this file with python (*python main.py*) for obtaining the its execution parameters. An usual parameters is *-full*, which inserts all the data: all ontology terms, genes, annotations, and import the experiments from all data sources configured in the file *src/data_sources.py*. Note that depending the imported data size, mainly the number of experiment and their sizes, the data import process can last days or even weeks. The *-full* parameter can be used for inserting new data as well. Other options, such as *-ontology*, *-gene_ontology*, *-annotations*, and *-datasets* are used for inserting or updating only the entities in their scope.

The *Populator* source code is organized as:

- data/ - Initial data used by the Populator
 - annotations/ - Annotations file (CpG Island, Regulatory Build, Repeat Regions)
 - cv/ - Controlled vocabulary files from ENCODE

¹ <http://www.lua.org/>

Category	Command	Description
Region	value_of(column_name)	Return the column value
String	string.find(s,motif)	Return the first match of motif in the string
	string.format(fmt,...)	Return a formatted string
	string.gmatch(s,motif)	Return the next captures from motif over string s
	string.len(s)	Return string length
	string.upper(s)	Return a copy of a string with all letters changed to uppercase
	string.lower(s)	Return a copy of a string with all letters changed to lowercase
	string.reverse(s)	Return a reverse copy of the string
	string.sub(s,i[,j])	Return a substring
Math	math.abs(x)	Return the absolute value of x
	math.ceil(x)	Return the smallest integer larger than or equal to x
	math.exp(x)	Return the value e^x
	math.floor(x)	Return the largest integer smaller than or equal to x
	math.fmod(x,y)	Return the remainder of the division of x by y
	math.log(x)	Return the natural logarithm of x
	math.log10(x)	Return the base-10 logarithm of x
	math.max(x,...)	Return the maximum value among its arguments
	math.min(x,...)	Return the minimum value among its arguments
	math.modf(x)	Return two numbers, the integral and the fractional part of x
	math.pi	Return the value of π
	math.pow(x,y)	Return x^y
	math.random([m,[n]])	Return a random integer in the range $[m, n]$
	math.sqrt(x)	Return the square root of x
Conversion	type(o)	the type of the variable
	tonumber(o)	Convert the value to a number
	tostring(o)	Convert a number to a string
OS calls	os.clock	
	os.difftime	
	os.time	
Debugging	print	Print variable for server debugging
	pcall	Call a function in protective mode for server debugging

Table A.10: Lua commands available in the *DeepBlue Server*.

- genomes/ - Genomic data (chromosome sizes and sequences)
- ontologies/ - Imported ontologies (*CL*, *EFO*, *UBERON*)
- src/ - Populator source code
 - download/ - Data downloaded by the Populator
- third_party/ - Third party tools

A.2.3 DeepBlueR

Installation

As the *DeepBlueR* is integrated in the *R/Bioconductor* environment, it can be installed in two commands: `source("https://bioconductor.org/biocLite.R")` and `biocLite("DeepBlueR")`. After the installation is completed, the *DeepBlueR* library is loaded through the command

`library(DeepBlueR)`, and it can be tested using the command `deepblue_info("me")`. This command returns a data structure containing information about the user, in this case, the *anonymous user*.

Implementation

DeepBlueR is implemented in *R* and *Python*. Where the *R* code is divided into the auto-generated and pure *R* source code. The pure *R* code contains examples, an optimized implementation of the XML-RPC protocol, data caching, data types conversion between XML-RPC and *R*, and requests splitting for improving the performance in genome-scale requests. The auto-generated source code contains the code that envelopes the *DeepBlue Server* operation requests, where each operation has its auxiliary code.

The *Python* source code automatically generates *R* code. It follows a workflow, where firstly obtains all available commands from the *DeepBlue Server* API and generates the appropriated *R* code for encapsulating their usage. The *R* code generation has the following steps:

1. Execute the operation *commands* for obtaining the *DeepBlue Server* API operations
2. Interpret API definitions and generate the *R* code that encapsules each command
3. Insert the previously developed code examples
4. Execute and test the examples
5. Build the documentation (Vignettes files)
6. Commit the changes

This workflow minimizes the chance of errors, performs automatic code verification and testing, and provides a faster release cycle, coping with the evolution of the *DeepBlue Server*.

A.2.4 DeepBlue Web Portal

The *DeepBlue Web Portal* is implemented using standard web tools technologies: *HTML*, *CSS*, *PHP* and *JavaScript*. It also uses the following external libraries:

- **SmartAdmin** is the templates content that was used as skeleton for the web portal.
- **Bootstrap** provides web components for constructing web interfaces - getbootstrap.com/
- **JQuery** provides auxiliary web components - jquery.com/
- **DataTables** provides data tables - datatables.net/
- **Fonts Awesome** provides icons - fontawesome.com
- **morris.js** provides charts - morrisjs.github.io/morris.js/
- **Sweet Alert** provides alert boxes - sweetalert.js.org/
- **Select2** provides data selection boxes - select2.org/
- **Bootstro.js** provides an online user guidance - clu3.github.io/bootstro.js
- **Incutio XML-RPC** provides the communication between the *PHP* code and the *DeepBlue Server* - scripts.incutio.com/xmlrpc

The Web Portal access the *DeepBlue Server* through its *PHP* and *JavaScript* code: the *PHP* code access *DeepBlue* using the XML-RPC protocol, while the *JavaScript* uses *RESTful* protocol with *AJAX*².

The *DeepBlue Server* API operation *datatable* was implemented for exclusively supporting the web portal. This operation was developed and tailored for being compatible with the *datatables* library. The *datatables* web interface sends their *RESTful* requests to the *DeepBlue Middleware*. The *DeepBlue Middleware* translates the requests to XML-RPC requests, which are sent to the *DeepBlue Server*. The *DeepBlue Server* processes these requests in real-time, providing fast replies to the interactions between users and the *DeepBlue Web Portal datatables*.

The *DeepBlue Web Portal* can be served by an *Apache HTTP* server with *PHP* installed. The *DeepBlue Web Portal* URL and the *DeepBlue Server* location must be informed in the configuration file *template/lib/server_settings.php*.

A.2.5 DeepBlue Middleware

The *DeepBlue Middleware* is a *DeepBlue Server* proxy. It is implemented in *JavaScript* and *TypeScript*³ and executed in the *Node.js*⁴ environment.

The *DeepBlue Middleware* was initially developed as data cache for improving the *Web Portal* response time and for converting *RESTful* to XML-RPC requests. Currently, it also contains the *DIVE* server-side functionalities, such as workflows orchestration (more in Section *app:dive:implementation*).

A.2.6 DIVE

DIVE has been developed using modern and open technologies, such as the *Angular Framework*⁵, used for web applications development, the *TypeScript* programming language, used in the *UI* and server, and the *Node.js* for executing the server applications. In this case, the *Node.js* is used by the *DeepBlue Middleware* that contains the *DIVE* server-side methods.

The *UI* web application is developed using *Angular 5*, *PrimeNG*⁶, *Highcharts*⁷, *ngx-datatable*⁸, *angular-archwizard*⁹, and *randomcolor*¹⁰.

DIVE implementation is divided into two parts: (i) the *UI* web implementation contains the source code of the application that is executed in the user web browsers; (ii) server implementation that extends the *DeepBlue Middleware* and is developed in *TypeScript*.

² <http://api.jquery.com/jquery.ajax/>

³ <https://www.typescriptlang.org/>

⁴ <https://nodejs.org>

⁵ <https://angular.io/>

⁶ <https://www.primefaces.org/primeng/>

⁷ <https://www.highcharts.com/>

⁸ <https://swimlane.github.io/ngx-datatable/>

⁹ <https://www.npmjs.com/package/angular-archwizard>

¹⁰ <https://github.com/davidmerfield/randomColor>

UI components

Epigenomic data analysis is changing quickly, with new assay types and data. Hence, it is fundamental to ease the future development of *DIVE*. For keeping in pace with new technologies, implementing new analysis and visualization methods, *DIVE* is developed using individual components. These individual components are used for constructing the entire *DIVE*'s UI, from buttons, menus, tables, charts, to groups of elements that interact together, to the complete interface.

DIVE UI elements are implemented as independent components and their source code are stored in the *src/app/view* directory. The advantages of having *DIVE*'s UI source code separated into different component are: (i) it improves the reuse of the component in different parts of *DIVE* or even in different applications; (ii) it allows independent modifications and testing of each component; (iii) it facilitates extending *DIVE* and future improvements.

In overall, *DIVE* components can be categorized as: (i) screens; (ii) main-screen components; (iii) in-context components. The following lists presents an overview of the *DIVE* UI components:

- *Screen components*: these are the backbones of the interface, where the UI is built upon. Two types of screen components are available:

Screens	are the component that connects all components for forming a complete interface
Wizards	provides interfaces that guide users through data selection and methods configuration options
- *Main-screen components* perform fundamental tasks:

Filtering Stack	is the stack where the current region-set is exhibited with its applied filters
Comparison Datasets	is the top menu which shows the comparison region-sets
Menu	is the menu, which loads its options dynamically from the existing datasets, such as, histone marks and CSS items
- *In-context components* perform special tasks:

Data Selection	are drop-boxes and lists for visualizing and selecting data
Charts	provides the framework for the overlapping and similar datasets charts

UI Services

DIVE contains different services (*src/app/service*) that perform and process requests to the *DeepBlue Middleware*, control the current status of the data presented in the UI, and perform mathematical operations. *DIVE* has services for the following tasks:

- request metadata information to the *DeepBlue Middleware*
- manage the processing requests made to the *DeepBlue Middleware*.
- control the selected data
- control the comparison data

- control filters values and apply on the data
- calculate statistical values

Server side processing

The *DeepBlue Middleware* (Appendix A.2.5) was initially developed as a middleware between the *DeepBlue Web Portal* and the *DeepBlue Server*. With the development of *DIVE*, it was perceived that this tool also needs functionalities that orchestrate the requests made to the *DeepBlue Server*.

The middleware was extended to provided functionalities such as the execution of hundreds to thousands of data operations in the *DeepBlue Server*. It orchestrates their executions and returns only the final results to the *DIVE UI*, thus, speeding up the overall execution through minimizing the number of *RESTful* calls between the web application and the *DeepBlue Server*.

The *DeepBlue Middleware* part that process *DIVE* requests is developed in *TypeScript*, which is compiled to *JavaScript*, and is included in the *DeepBlue Middleware* codebase, which is executed using *Node.js*.

Concluding, *DIVE*'s implementation allows easy extendability of its functionalities and optimization of the existing ones for future changes in the epigenomic data landscape. Finally, besides being a data analysis tool, it is a framework which allows the development of new functionalities, aligning to the future needs of the epigenomic large-scale data analysis.

A.3 Usage example and use cases source code

A.3.1 DeepBlue Server

Listing A.1 defines a function named `__wait_and_get_data` that queries the *DeepBlue Server* for information in the given *request_id* each second until its processing is done or failed, then it returns its result.

```

1 import xmlrpclib
2 import time
3
4 DEEPBLUE_URL = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
5
6 # Wait for the server processing and return the data
7 def __wait_and_get_data(request_id, user_key):
8
9     server = xmlrpclib.Server(DEEPBLUE_URL, allow_none=True)
10    (status, info) = server.info(request_id, user_key)
11    request_status = info[0]["state"]
12
13    while request_status != "done" and request_status != "failed":
14        time.sleep(1)
15        (status, info) = server.info(request_id, user_key)
16        request_status = info[0]["state"]
17
18    return server.get_request_data(request_id, user_key)

```

Listing A.1: Auxiliar function `__wait_and_get_data` which waits for the workflow processing finishes and then returns the resulting data. This function is used in the usage examples for downloading the workflow results.

Usage Example 1: Identification of TFBSs that overlap *H3K4me3* peaks and promoter regions

Listing A.2 source code shows how to obtain the active TFBSs by overlapping a set of TFBSs to *H3K4me3* peaks, an active mark, and after, to the dynamic generate promoter regions. It provides a list of TFBSs annotated with the experiment source and biosource that match the previous criteria.

```

1  import xmlrpclib
2  import time
3  from pprint import pprint
4
5  url = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
6  user_key = "anonymous_key"
7
8  server = xmlrpclib.Server(url, allow_none=True)
9
10 # Select all peaks regions from H3k37ac from BLUEPRINT
11 (status, exps) = server.select_regions("", "GRCh38", "H3K4me3", "", "",
12                                     "BLUEPRINT Epigenome", "chr1",
13                                     None, None, user_key)
14 (status, exps_peaks) = server.query_experiment_type(exps, "peaks", user_key)
15
16 # Select promoters
17 (status, q_genes) = server.select_genes(
18     None, None, "gencode v23", None, None, None, user_key)
19 (s, promoters) = server.flank(q_genes, -2500, 2000, True, user_key)
20
21 # Intersect the H3K4me3 peaks with the gene promoters.
22 (status, exps_promoters) = server.intersection(exps_peaks, promoters, user_key)
23
24 # Select the SP1 peaks regions from ENCODE
25 (status, tf) = server.select_regions("", "hg19", [
26     "SP1"], "", "", "ENCODE", "chr1", None, None, user_key)
27 (status, ts_signals) = server.query_experiment_type(tf, "peaks", user_key)
28
29 # Intersect the SP1 regions with the H3K4me3 peaks that overlaps with promoters.
30 (status, final) = server.intersection(tf, exps_promoters, user_key)
31
32 # Obtain and annotate regions with the
33 # experiment name (@NAME) and Biosource (@BIOSOURCE).
34 (status, request_id) = server.get_regions(final,
35                                     "CHROMOSOME,START,END,@NAME,@BIOSOURCE",
36                                     user_key)
37
38 regions = __wait_and_get_data(request_id, user_key)

```

Listing A.2: This source code shows how to obtain the active TFBSs by overlapping a set of TFBSs to *H3K4me3* peaks, an active mark, and after, to the dynamic generate promoter regions: Lines 1 - 8 import the necessary libraries and assign to the variable `server` to a XML-RPC object for accessing the *DeepBlue Server*. Lines 10 - 13 selects all the *H3K4me3* from the BLUEPRINT project. Lines 17 - 19 select the genes from the Gene Model GENCODE v23 and generate the promoter regions. Line 22 filters the *H3K4me3* regions that overlap with a promoter region. Lines 25 - 27 selects the TFs *SP1* from ENCODE project. Line 30 filters the TFs regions that overlap with the previously filtered *H3K4me3* region. The lines 34 - 36 requests the regions, where each region contains its chromosome, start, end, experiment name, epigenetic mark, and biosource. Line 38 download the regions data.

Listing A.3 summarizes DNA methylation levels in liver tissue across *H3K4me3* peaks regions derived from human embryonic stem cells. It generates a list of files which contains the DNA methylation summarized for each found liver tissue signal experiment.

[illegible]

```
61
62     # Request regions
63     status, req = server.get_regions(q_filter,
64                                     "CHROMOSOME,START,END,@AGG.MEAN,@AGG.COUNT,@AGG.MAX,@AGG.MIN", user_key)
65     print liver_experiment, status, req
66     request_id_experiment[req] = liver_experiment
67     requests.append(req)
68
69 if not os.path.isdir("data/"):
70     os.mkdir("data/")
71
72 # Waiting for the workflow processed requests be finished
73 # Store results in individual files and print the request info in case of error
74 while len(requests) > 0:
75     for req in requests[:]:
76         (s, ss) = server.info(req, user_key)
77         if ss[0]["state"] == "done":
78             print ss[0]
79             print "getting data from " + ss[0]["_id"]
80             (s, data) = server.get_request_data(req, user_key)
81             with open("data/"+request_id_experiment[req]+".bed", 'wb') as f:
82                 f.write(data)
83             requests.remove(req)
84         if ss[0]["state"] == "failed":
85             print ss
86             requests.remove(req)
87     time.sleep(1.0)
```


Listing A.3: Processing flow of this usage example: (i) list and select the regions from *H3K4me3* and *ESCs* experiments, these regions are used as summarizing boundaries; (ii) list all liver and hepatocyte experiments data; (ii) for each experiment, summarize it using the previously obtained *H3K4me3* peaks and store the data in a file: Lines 1 - 7 import the necessary libraries and assigns the variable *server* to an XML-RPC object for accessing the *DeepBlue Server*. Line 8 tests the connection to the server with the operation *echo*. Lines 10 - 13 list and extract all samples *IDs* with the biosource *H1-hESC* from the *ENCODE* project. Line 16 lists all peaks experiments that contain the previously selected samples *IDs*, the histone modification *H3K4me3* from the *ENCODE* project. Lines 21 - 24 extract the *IDs* from the listed experiments, obtain information about the experiment using the *ID*, and generate a list of experiments that the original file name ends with *bed.gz*. Lines 25 - 26 verify if it found only one experiment file because it requires only one experiment regions for summarizing the data. Line 30 selects the regions of the selected *h1-hESC H3K4me3* experiment. Lines 35 - 40 list and extract the names of the DNA methylation experiments that are annotated with liver or hepatocyte biosource. Lines 47 - 52 iterate over each selected DNA methylation experiment, selecting its regions. Lines 55 - 56 aggregate the selected regions using *H1-hESC* regions. It performs the aggregation on the column *SCORE*. Lines 59 - 60 filters and remove the aggregated regions that did not aggregate any region. Lines 63 - 67 request the regions with the desired columns. It stores the experiment name with the associated request *ID*, and also the request *ID* in a list of *IDs*. Lines 69 - 70 create a directory to store the data downloaded. Lines 74 - 87 download the data, checking each request status: if the request is done, its data is downloaded, stored in a file, and the request is removed from the requests list. It is repeated until all requests are processed.

Usage Example 3: Enriching DMR regions via Chromatin States

Listing A.4 enriches *DMR* regions using chromatin states regions dynamically obtained from all *CSS* files in the genome *GRCh38*. This example is divided into two parts: constructing the *CSS* region-sets, selecting and filtering the hypomethylated regions, and performing the enrichment on them.

```

1 import xmlrpclib
2 import time
3 import os.path
4 import errno
5
6 from collections import defaultdict
7 from pprint import pprint
8 from multiprocessing import Pool
9 from socket import error as socket_error
10
11 import cPickle
12
13 # Internal configurations
14 MAX_REQUEST_SIMULTANEOUS = 16

```

```

15 CACHE_PATH = ".cache"
16 STATES_CACHE_FILE = "css_queries_cache.deepblue"
17 DEEPBLUE_URL = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
18 USER_KEY = 'anonymous_key'
19
20 GENOME = 'GRCh38'
21 # Experiment that will be enriched
22 EXPERIMENT_NAME = "S00VEQA1.hypo_meth.bs_call.GRCh38.20150707.bed"
23
24
25 def split_file(data):
26     server_worker = xmlrpc.Server(DEEPBLUE_URL, allow_none=True)
27
28     [css_file, states] = data
29     result = []
30
31     _, css_file_query_id = server_worker.select_experiments(
32         css_file, None, None, None, USER_KEY)
33     # Use query cache for optimizing the processing because the same experiment
34     # data will be used many times.
35     _, css_file_query_cache_id = server_worker.query_cache(
36         css_file_query_id, True, USER_KEY)
37     for state in states:
38         _, css_file_filter_query_id = server_worker.filter_regions(
39             css_file_query_cache_id, "NAME", "==", state, "string", USER_KEY)
40         result.append((state, css_file_filter_query_id))
41
42     return css_file, result
43
44
45 def get_chromatin_states(genome, user_key):
46     status, csss_query_id = SERVER.select_regions(
47         None, genome, "Chromatin State Segmentation",
48         None, None, None, None, None, None, user_key)
49
50     status, csss_names_request_id = SERVER.distinct_column_values(
51         csss_query_id, "NAME", user_key)
52     distinct_values = __wait_and_get_data(csss_names_request_id)
53
54     return distinct_values['distinct']
55
56
57 def build_chromatin_state_files(server, genome, user_key):
58     cache_file = os.path.join(CACHE_PATH, STATES_CACHE_FILE)
59
60     if os.path.exists(cache_file):
61         _file = open(cache_file, "r")
62         queries = cPickle.load(_file)
63         return queries
64     else:
65         _, css_files_list = server.list_experiments(
66             genome, "peaks", "Chromatin State Segmentation", None, None, None, None, user_key
67             )
68         _, css_files_names = server.extract_names(css_files_list)
69         states = get_chromatin_states(genome, user_key)
70
71         css_files = []
72         for css_file in css_files_names:
73             css_files.append([css_file, states])
74
75         pool = Pool(MAX_REQUEST_SIMULTANEOUS)
76         queries = pool.map(split_file, css_files)
77         pprint(queries)
78         datasets = defaultdict(list)
79
80         for query in queries:
81             for state in query[1]:

```

```

81         datasets[query[0]].append([state[1], state[0]])
82
83     datasets = dict(datasets)
84
85     pool.close()
86     pool.join()
87
88     # Save the processed states in a file.
89     try:
90         os.makedirs(CACHE_PATH)
91     except OSError as exc: # Python >2.5
92         if exc.errno == errno.EEXIST and os.path.isdir(CACHE_PATH):
93             pass
94         else:
95             raise
96     _file = open(cache_file, "w+")
97     cPickle.dump(datasets, _file)
98
99     return datasets
100
101
102 SERVER = xmlrpclib.Server(DEEPBLUE_URL, allow_none=True)
103
104 # Query data
105 # Select the hypo methylated regions where the AVG methyl level is lower than 0.0025
106 _, QUERY_ID = SERVER.select_experiments(EXPERIMENT_NAME, None, None, None, USER_KEY)
107 _, QUERY_ID = SERVER.filter_regions(
108     QUERY_ID, "AVG_METHYL_LEVEL", "<", "0.0025", "number", USER_KEY)
109 print QUERY_ID
110
111 # Universe will be tiling regions of 1000bp
112 _, UNIVERSE_QUERY_ID = SERVER.tiling_regions(1000, "grch38", None, USER_KEY)
113
114 # Datasets will be the chromatin segmentation files divided by segments
115 DATASETS = build_chromatin_state_files(SERVER, GENOME, USER_KEY)
116 print "total of " + str(len(DATASETS)) + " datasets"
117
118 # Obtain the data
119 __, ENRICH_REQUEST = SERVER.enrich_regions_overlap(
120     QUERY_ID, UNIVERSE_QUERY_ID, DATASETS, "grch38", USER_KEY)
121 print "Processing enrich_region_overlap: ", ENRICH_REQUEST
122
123 ENRICHMENT_RESULT = __wait_and_get_data(ENRICH_REQUEST)
124
125 # Obtain the ranks
126 state_rank = [(k["description"], k["mean_rank"])
127               for k in ENRICHMENT_RESULT["enrichment"]["results"]]
128 pprint(state_rank)
129
130 # ('12_Active_TSS_High_Signal_H3K4me3_H3K27Ac', 30.6667),
131 # ('12_Active_TSS_High_Signal_H3K4me3_H3K27Ac', 33.3333),
132 # ('12_Active_TSS_High_Signal_H3K4me3_H3K27Ac', 37.6667),
133 # ('12_Active_TSS_High_Signal_H3K4me3_H3K27Ac', 40.3333),
134 # ('12_Active_TSS_High_Signal_H3K4me3_H3K27Ac', 40.6667), ...

```

Listing A.4: It enriches *DMR* regions using chromatin states regions dynamically obtained from all *CSS* files in the genome *GRCh38*. This example is divided into two parts: constructing the *CSS* region-sets, selecting and filtering the hypomethylated regions, and performing the enrichment on them. Lines 1 to 22 load the necessary libraries and set-up the variables names containing the parameters used during the processing. Line 25 - 42 contain the function *split_file* which receives a *CSS* file name and split it into multiple request *IDs*, each one referencing to chromatin state and a sub-set of the file. Line 45 - 54 obtains all chromatin states presents in the genome *CSS* files. Line 57 - 99 build divides all *CSS* files presents in the genome by their states, generating $n \times m$ region-sets (m is the count of available *CSS* experiments and n is the count of chromatin states). Line 97 uses the *cPickle* library for storing the *CSS* region-sets locally for improving the next executions of this script. (The stored data is loaded in line 62). Line 115 stores the region-sets in the variable *DATASETS* that is used in line 119 by the *enrich_regions_overlap* operation. This operation uses the variable *QUERY_ID* which is the hypomethylated experiment filtered in lines 107 - 108. Line 123 waits for the enrichment processing and downloads its result. Lines 126 - 128 obtain the enrichment results, sort them by their mean rank and print the sorted result.

Usage Example 4: Obtaining (epi)genomic information from different tissues

Listing A.5 provides data for comparing the *FAR1* gene epigenomic information in the liver and brain tissues. For that, it obtains the gene genomic location and generates flanking regions, for promoters and region after the gene. Using them for summarizing the different epigenetic marks data. As result, it provides a set of files, each of a different epigenetic mark, containing the epigenetic mark data summarized by the defined genomic regions.

```

1 import re
2
3 from collections import defaultdict
4
5 DEEPBLUE_URL = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
6 USER_KEY = 'anonymous_key'
7
8 server = xmlrpclib.Server(DEEPBLUE_URL, allow_none=True)
9
10 # Select gene and build the surrounding regions.
11 # Select the gene region and also the promoter
12 status, q_genes = server.select_genes(
13     "FAR1", None, "gencode v19", None, None, None, USER_KEY)
14 # If error, print the error message
15 if status != "okay":
16     print q_genes
17 (s, before_flank_id) = server.flank(q_genes, -2500, 2000, True, USER_KEY)
18 (s, after_flank_id) = server.flank(q_genes, 1500, 500, True, USER_KEY)
19 (s, flank_merge_id) = server.merge_queries(
20     before_flank_id, after_flank_id, USER_KEY)
21 (s, q_genes_and_promoters) = server.merge_queries(
22     q_genes, flank_merge_id, USER_KEY)
23
24
```

```

25 # Data of interest
26 epi_marks = ["H3K36me3", "H3K4me3", "H3K27me3", "H3K4me1", "H3K9me3",
27             "H3K27ac", "DNA Methylation", "DNaseI", "RNA"]
28 biosources = ["brain", "liver"]
29 data_type = "signal"
30 project = "Roadmap Epigenomics"
31 genome = "hg19"
32
33
34 # Manually selected these Epigenomes from Roadmap
35 roadmap_samples = set(['E071', 'E054', 'E070', 'E053', 'E082', 'E066', 'E118'])
36
37 em_by_sample = defaultdict(list)
38 sample_em_exp = defaultdict(lambda: defaultdict(list))
39 experiment_biosource = {}
40
41
42 for epigenetic_mark in epi_marks:
43     em_name = None
44     for biosource in biosources:
45         status, exps = server.list_experiments(
46             genome, data_type, epigenetic_mark, biosource, None, None, project, USER_KEY)
47         exps.sort()
48         for exp in exps:
49             exp_name = exp[1]
50
51             (exp_id, em) = re.split("[-.]", exp_name, 1)
52             (em_name, compl) = re.split("\.", em, 1)
53             if epigenetic_mark == "RNA":
54                 em_name = "RNA_"+compl
55
56             experiment_biosource[exp_id] = biosource
57             em_by_sample[em_name].append(exp_id)
58             sample_em_exp[em_name][exp_id] = exp_name
59         if em_name:
60             em_by_sample[em_name] = set(
61                 em_by_sample[em_name]).intersection(roadmap_samples)
62         else:
63             print 'missing', epigenetic_mark
64
65 for epigenetic_mark in em_by_sample:
66     samples = em_by_sample[epigenetic_mark]
67
68     experiments = []
69     for sample in samples:
70         experiment_name = sample_em_exp[epigenetic_mark][sample]
71         experiments.append(experiment_name)
72
73     experiments_columns = {}
74     for experiment_name in experiments:
75         experiments_columns[experiment_name] = "VALUE"
76
77     (status, request_id) = server.score_matrix(
78         experiments_columns, "mean", q_genes_and_promoters, USER_KEY)
79     data = __wait_and_get_data(request_id, USER_KEY)
80     f = open("__"+epigenetic_mark+".csv", "w+")
81     f.write(data)
82     f.close()

```

Listing A.5: It can be divided in 4 parts: Lines 1 - 8 load the libraries and defining variables for accessing the *DeepBlue Sever*. Lines 12 - 22 load the gene *FAR1* location and define surrounding regions. Lines 26 - 35 define the data of interest. Lines 42 - 63 obtain the samples and experiments of the selected BioSources. Lines 65 - 82 obtain the data from the experiments, summarizing them by the gene body and surrounding regions, and writing the results in a file, where each file contains the data of an epigenetic mark.

Usage Example 5: Summarize gene expression from hepatocytes experiments

Listing A.6 summarizes the expression of a set of pre-selected genes using different hepatocytes samples from the CREST project. It presents the summarization result in the form of box plot.

```

1 import matplotlib.pyplot as plt
2 from io import BytesIO
3 import numpy as np
4 import time
5 import xmlrpclib
6
7 url = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
8 deepblue = xmlrpclib.Server(url, allow_none=True)
9
10 USER_KEY = "anonymous_key"
11
12 # List all mRNA experiments from Hepatocytes and from the CREST project.
13 status, experiments = deepblue.list_experiments("GRCh38", "signal",
14                                                  "mRNA", "hepatocyte",
15                                                  None, None, "CREST",
16                                                  USER_KEY)
17
18 # Extract the names from the ID,name pair
19 status, experiment_names = deepblue.extract_names(experiments)
20
21 # Some genes manually selected.
22 # These genes are active in the hepatocytes
23 genes = ["ADH1A", "ADH1C", "ADH4", "ADH5", "ADH6", "ADH7",
24          "GSTA1", "GSTA2", "GSTA3", "GSTA4"]
25
26 (status, q_genes) = deepblue.select_genes(genes, None,
27                                           "gencode v23", None, None, None, USER_KEY)
28
29 # Obtain the genes from the server.
30 # It is necessary to obtain the genes in "genomic order".
31 # The gene names are obtained using the the meta-field @GENE_NAME
32 (status, request_id) = deepblue.get_regions(q_genes,
33                                           "@GENE_NAME(gencode v22)",
34                                           USER_KEY)
35
36 genes_ordered = __wait_and_get_data(request_id)
37
38 # Build score matrix
39
40 # Create a dictionary, with the experiment names as key and the column name as value.
41 # For this case, it uses the column named "VALUE" that is the default value column in the
42   DeepBlue signal data.
43 experiments_columns = {}
44 for experiment_name in experiment_names:
45     experiments_columns[experiment_name] = "VALUE"

```

```

46 # Build the score_matrix using the defined experiments_columns where it calculates the mean
    value by the regions defined in the q_genes.
47 status, score_matrix_request = deepblue.score_matrix(experiments_columns,
48                                                         "mean",
49                                                         q_genes,
50                                                         USER_KEY)
51 # Download the data
52 score_matrix_genes = __wait_and_get_data(score_matrix_request)
53
54
55 # Importing the data into numpy
56 # Import the data into a numpy data structure.
57 # Use tabs as delimiters, skipping the first row (the header), and using columns from the 3rd
    position (chr, start, end)
58 experiments_count = len(experiment_names)
59 data = np.genfromtxt(BytesIO(score_matrix_genes), delimiter="\t",
60                       skip_header=1,
61                       usecols=range(3, experiments_count+3))
62
63
64 # Plotting the data
65 # Use the matplotlib for generating a boxplot of the gene expressions.
66 fig = plt.figure(1, figsize=(14, 10))
67 plt.xlabel('Gene')
68 plt.ylabel('Expression level')
69
70 # Create an axes instance
71 ax = fig.add_subplot(111)
72
73 # Create the boxplot
74 bp = ax.boxplot(data.transpose(),
75                 labels=genes_ordered.split("\n"))
76
77 plt.show()

```

Listing A.6: Summarizes the expression of a set of pre-selected genes using different hepatocytes samples from the CREST project and presents the result in the form of box plot. Lines 1 to 10 load the libraries define the variables for accessing the *DeepBlue Server*. Lines 13-14 list and extract the names of all *mRNA* hepatocyte experiments of the CREST project. Lines 22 - 36 defines the genes which are analyzed, load their information, and obtain their list from the *DeepBlue Server* ordered by their genomic location. Lines 42 - 52 obtain a score matrix containing the genes *mRNA* expression levels. Lines 58 - 61 load the score matrix data into a *NumPy* matrix that is used in lines 66 - 77 for plotting a bar chart that is exhibited to the user.

A.3.2 DeepBlueR

Listing A.7 presents the summarization and clustering of the DNA methylation data level across 206 BLUEPRINT blood samples. Displaying a heatmap to the user.

Blueprint methylation

```

1 library(DeepBlueR)
2 library(foreach)
3 library(stringr)
4 library(matrixStats)
5 library(gplots)
6 library(RColorBrewer)

```



```

7
8 ## 1) Select data and columns
9 # Select experiments
10 blueprint_DNA_meth <- deepblue_list_experiments(genome = "GRCh38",
11                                                epigenetic_mark = "DNA Methylation",
12                                                technique = "Bisulfite-Seq",
13                                                project = "BLUEPRINT EPIGENOME")
14
15 # Filtering for call files (remove the coverage files)
16 blueprint_DNA_meth <- blueprint_DNA_meth[grepl("CPG_methylation_calls.bs_call",
17        deepblue_extract_names(blueprint_DNA_meth)),]
18
19 # Define the colum (VALUE) that will be used for summarization
20 exp_columns <- deepblue_select_column(blueprint_DNA_meth, "VALUE")
21
22
23 ## 2) Select individually the chromosomes of the anotation Blueprint Ensembl Regulatory Build
24 # list all available chromosomes in GRCh38
25 chromosomes_GRCh38 <- deepblue_extract_ids(
26     deepblue_chromosomes(genome = "GRCh38")
27 )
28 # keep only the essential ones
29 chromosomes_GRCh38 <-
30     grepl(pattern = "chr([0-9]{1,2}|X)$", chromosomes_GRCh38, value = TRUE)
31
32 # Split the request by chromosome to avoid hitting the memory limit of DeepBlue
33 blueprint_regulatory_regions <-
34     foreach(chr = chromosomes_GRCh38, .combine = c) %do%
35         deepblue_select_annotations(
36             annotation_name = "Blueprint Ensembl Regulatory Build",
37             chromosome = chr,
38             genome = "GRCh38"
39         )
40
41 ## 3) Generate the score matrix
42 # Request the processing of score matrices
43 request_ids <- foreach(query_id = blueprint_regulatory_regions,
44                       .combine = c) %do%
45     deepblue_score_matrix(
46         experiments_columns = exp_columns,
47         aggregation_function = "mean",
48         aggregation_regions_id = query_id)
49
50 # Obtain the score matrix results
51 score_matrix <- data.table::rbindlist(
52     deepblue_batch_export_results(request_ids),
53     use.names = TRUE)
54
55 ## 4) Processing the input data
56 # Remove the first three columns (CHROMOSOME, START, END) and convert the data frame to a
57   numeric matrix.
58 filtered_score_matrix <- as.matrix(score_matrix[,-c(1:3), with=FALSE])
59
60 # Compute the variance of each row and retain only genomic regions with variance > 0.05 for
61   plotting
62 filtered_score_matrix_rowVars <- rowVars(filtered_score_matrix, na.rm = TRUE)
63 filtered_score_matrix <- filtered_score_matrix[which(filtered_score_matrix_rowVars > 0.05),]
64
65 # To be able to cluster samples, remove regions that have missing values in at least one of
66   the experiments.
67 filtered_score_matrix <- filtered_score_matrix[which(complete.cases(filtered_score_matrix)),]
68
69 # It is collected the metadata for each experiment
70 experiments_info <- deepblue_info(deepblue_extract_ids(blueprint_DNA_meth))
71
72 # Obtain the experiment names
73 exp_names <- unlist(lapply(experiments_info, function(x){ x$name}))

```

```

71
72 # IMPORTANT: The order of columns in the score matrix is not the same as in the exp_columns
    list used in the request. It has to order the matrix by the experiment names in the
    color map. This is crucial to make sure it assigns the correct cell type to each sample.
73 filtered_score_matrix <- filtered_score_matrix[,exp_names]
74
75 ## 5) Generating a heatmap
76 # Metadata and colors
77 getPalette <- colorRampPalette(brewer.pal(9, "Set1"))
78
79 # Obtain the biosource names
80 biosource <- unlist(lapply(experiments_info, function(x){ x$sample_info$biosource_name}))
81
82 # Replace positive with + and negative with - for saving space
83 biosource <- str_replace_all(biosource, "-positive", "+")
84 biosource <- str_replace_all(biosource, "-negative", "-")
85
86 # For same reason, the ', terminally differentiated' text is removed
87 biosource <- str_replace(biosource, ", terminally differentiated", "")
88
89 # Assigning unique color to each BioSource
90 color_map <- data.frame(biosource = unique(biosource),
91                        color = getPalette(length(unique(biosource))))
92
93 # Assign colors to the experiments
94 biosource_colors <- data.frame(name = exp_names, biosource = biosource)
95 biosource_colors <- dplyr::left_join(biosource_colors, color_map, by = "biosource")
96 # Transform this data frame into a vector that is compatible with the heatmap function.
97 color_vector <- as.character(biosource_colors$color)
98 names(color_vector) <- biosource_colors$biosource
99
100 ## 6) Plotting
101 # It plots a heatmap in which the variable regions are shown across all samples.
102 # On top of the columns, it creates a dendrogram based on Pearson correlation.
103 heatmap.2(filtered_score_matrix, labRow = NA, labCol = NA,
104           trace = "none", ColSideColors = color_vector,
105           hclust=function(x) hclust(x,method="complete"),
106           distfun=function(x) as.dist(1-cor(t(x), method = "pearson")),
107           Rowv = TRUE, dendrogram = "column",
108           key.xlab = "beta value", denscol = "black", keysize = 1.5,
109           key.title = NA)
110
111 plot.new()
112
113 # Plot the legend
114 legend(x = 0, y = 1,
115        legend = color_map$biosource,
116        col = as.character(color_map$color),
117        text.width = 0.6,
118        lty= 1,
119        lwd = 6,
120        cex = 0.7,
121        y.intersp = 0.7,
122        x.intersp = 0.7,
123        inset=c(-0.21,-0.11))

```

Listing A.7: Summarization and clustering of the DNA methylation data level across 206 BLUEPRINT blood samples. Returning a heatmap to the user. Lines 1 - 6 load the required libraries. Line 10 obtains a list of all DNA methylation files provides by the BLUEPRINT project. This list is filtered in lines 16-17 for selecting only the methylation levels experiment files. Lines 25 - 29 build the list of chromosomes that are used in the analysis. Lines 33 - 39 selects the *BLUEPRINT ensembl regulatory build* that is used for the data aggregation. Lines 43 to 53 request the score matrix. The requests are divided by chromosome for improving the processing time through the data processing parallelization that is provided by the *DeepBlue Server*. Lines 57 - 64 filter the score matrix, removing the rows with empty values and with low variance. Lines 70 - 98 build the colors map used by the heatmap. Lines 103 - 123 plot the heatmap.

Predicting gene expression from histone marks

This use case predicts gene expression based on a set of histone marks. Due to its complexity, it is divided into two files: Listing A.8 load the data from the *DeepBlue Server* and generate the data file that is used by Listing A.9 which execute the predictions using the *glmnet* library and plot the charts containing the prediction results.

```

1 library(DeepBlueR)
2 library(data.table)
3 library(foreach)
4 library(GenomicRanges)
5 library(iterators)
6 library(dplyr)
7 library(tidyr)
8 library(stringr)
9
10 load(file = "matched_gene_expression_and_histone_data.RData")
11
12 #get histone data for the following histones
13 histone_marks <- c("H3K4me3", "H3K36me3", "H3K27ac", "H3K4me1", "H3K27me3", "H3K9me3")
14
15 #use specific biosource
16 selected_biosource = "CD4-positive, alpha-beta T cell"
17
18 #find out which samples have both gene expression and histone data
19
20 #samples with expression data
21 blueprint_samples_with_expr <- deepblue_info(deepblue_list_expressions(
22   project = "BLUEPRINT Epigenome",
23   expression_type = "gene")$id)
24
25 #get donor ids
26 blueprint_donors_with_expr <- unlist(lapply(blueprint_samples_with_expr,
27   function(x){x$sample_info$DONOR_ID}))
28
29 #experiments with histone data (optionally filter for specific cell type)
30 blueprint_experiments_with_histone <- deepblue_info(deepblue_extract_ids(
31   deepblue_list_experiments(genome = "GRCh38",
32     type = "signal",
33     project = "BLUEPRINT Epigenome",
34     epigenetic_mark = histone_marks,
35     biosource = selected_biosource)))
36

```

```

37 #collect some info on these experiments
38 blueprint_donors_with_histone <- rbindlist(
39   lapply(blueprint_experiments_with_histone,
40     function(x){data.frame(experiment_name = x$name,
41       sample = x$sample_id,
42       donor = x$sample_info$DONOR_ID,
43       histone_mark = x$epigenetic_mark,
44       biosource = x$sample_info$biosource_name)}}))
45
46 #keep experiments with data on all six histone marks on a single donor/biosource
47 #combo.
48 blueprint_donors_with_all_histone_marks <- blueprint_donors_with_histone %>%
49   group_by(donor, biosource) %>% filter(n() == 6)
50
51 #which donors have both expression data and data for six histone marks
52 donors_with_histone_data_and_gene_expr_data <- intersect(
53   blueprint_donors_with_expr, blueprint_donors_with_all_histone_marks$donor)
54
55 #subset histone meta data for donors that have also expression data
56 blueprint_all_histone_marks_and_expr <-
57   dplyr::filter(blueprint_donors_with_histone,
58     donor %in% donors_with_histone_data_and_gene_expr_data)
59
60 #build metadata for gene expression data
61 blueprint_gene_expr <- rbindlist(lapply(blueprint_samples_with_expr[
62   which(blueprint_donors_with_expr %in%
63     donors_with_histone_data_and_gene_expr_data)], function(x){
64     data.frame(exprs_id = x$id, exprs_sample = x$sample_id,
65       donor = x$sample_info$DONOR_ID,
66       biosource = x$sample_info$biosource_name)
67   })) %>% filter(biosource == selected_biosource)
68
69 #list all protein coding genes. Could also select genes differently or provide
70 #a user-defined list.
71 all_genes <- deepblue_list_genes(gene_model = "gencode v22")
72
73 protein_coding_genes <- all_genes[gene_type == "protein_coding" & source == "HAVANA"]
74 protein_coding_genes_id <- protein_coding_genes$gene_id
75 protein_coding_genes_name <- protein_coding_genes$gene_name
76
77 protein_coding_genes_query <- deepblue_select_genes(
78   genes = protein_coding_genes_name,
79   gene_model = "gencode v22")
80
81 #request gene expression data from DeepBlue
82 gene_expr_data_request_ids <- foreach(gene_expr_sample = as.character(
83   blueprint_gene_expr$exprs_sample),
84   .inorder = TRUE,
85   .final = function(x)
86     setNames(x, as.character(
87       blueprint_gene_expr$exprs_sample))) %do% {
88
89   blueprint_gene_expr_query <- deepblue_select_expressions(
90     expression_type = "gene",
91     sample_ids = gene_expr_sample,
92     identifiers = protein_coding_genes_id,
93     gene_model = "gencode v22")
94
95   blueprint_gene_expr_request_id <- deepblue_get_regions(
96     query_id = blueprint_gene_expr_query,
97     output_format = "CHROMOSOME,START,END,@STRAND(gencode v22),@GENE_NAME(gencode v22),
98       TPM")
99   return(blueprint_gene_expr_request_id)
100 }
101
102 #prepare getting histone data

```

```

103
104 #define promoter region
105 promoter_window <- deepblue_flank(protein_coding_genes_query, -2500, 5000)
106 #promoter_window <- deepblue_select_annotations("promoters", "GRCh38")
107
108 #general tiling regions of 100 bps
109 tiling_regions <- deepblue_tiling_regions(size = 100,
110                                           genome = "GRCh38")
111
112 #tiles in the promoter region
113 tss_tiling_regions <- deepblue_intersection(query_data_id = tiling_regions,
114                                           query_filter_id = promoter_window)
115
116 #request data for each histone mark
117 histone_request_ids <- foreach(hmark = histone_marks, .inorder = TRUE) %do%{
118
119   #filter for experiments for the current histone mark
120   blueprint_histone <- blueprint_all_histone_marks_and_expr %>%
121     dplyr::filter(histone_mark == str_to_lower(hmark))
122
123   #get request ids from DeepBlue
124   histone_data_request_ids <- foreach(histone_exp = as.character(
125     blueprint_histone$experiment_name),
126     .inorder = TRUE,
127     .final = function(x)
128       setNames(x, as.character(
129         blueprint_histone$experiment_name))) %do% {
130
131     data_id <- deepblue_select_experiments(
132       experiment_name=histone_exp)
133
134     query_id <- deepblue_aggregate(
135       data_id = data_id,
136       ranges_id = tss_tiling_regions,
137       column = "VALUE")
138
139     request_id <- deepblue_get_regions(
140       query_id, "CHROMOSOME, START, END, @AGG.MEAN")
141
142     return(request_id)
143   }
144 }
145 names(histone_request_ids) <- histone_marks
146
147 #check DeepBlue progress for gene expression data
148 if(any(deepblue_info(unlist(gene_expr_data_request_ids))$state %in% c("failed", "error"))){
149   stop("at least one gene expression data request failed")
150 } else if(any(deepblue_info(unlist(gene_expr_data_request_ids))$state != "done")){
151   stop("at least one gene expression data request is not finished yet")
152 } else{
153   message("gene expression data requests were processed successfully")
154 }
155
156
157 #download gene expression data and assign strand
158 gene_expr_data <- foreach(rid = gene_expr_data_request_ids,
159                           .inorder = TRUE,
160                           .final = function(x)
161                             setNames(x, names(gene_expr_data_request_ids))) %do%
162   {
163     regions_data <- deepblue_download_request_data(rid)
164     strand(regions_data) <- regions_data$`@STRAND(gencode v22)`
165     return(regions_data)
166   }
167
168 #check DeepBlue progress for each histone mark
169 foreach(hmark = histone_marks, .inorder = TRUE) %do%{

```

```

170
171 current_state <- deepblue_info(unlist(histone_request_ids[[hmark]]))$state
172
173 if(any(current_state %in% c("failed", "error"))){
174   stop(paste0("at least one gene expression data request failed for ", hmark))
175 }
176 else if(any(current_state != "done")){
177   stop(paste0("at least one gene expression data request is not finished yet for ", hmark))
178 }
179 else{
180   message(paste0("histone data requests were processed successfully for ", hmark))
181 }
182
183 }
184
185 #download and process data for each histone mark
186 histone_data <- foreach(hmark = histone_marks, .inorder = TRUE) %do%{
187
188   #filter for experiments for the current histone mark
189   blueprint_histone <- blueprint_all_histone_marks_and_expr %>%
190     dplyr::filter(histone_mark == str_to_lower(hmark))
191
192   #get request ids from above
193   histone_data_request_ids <- histone_request_ids[[hmark]]
194
195   #combine with gene expression meta data to obtain correct matches
196   #note that in case of several biological replicates all possible
197   #combinations are included.
198   metadata <- dplyr::inner_join(blueprint_histone, blueprint_gene_expr,
199     by = c("donor", "biosource"))
200
201   #download results
202   histone_tile_data <- foreach(rid = histone_data_request_ids,
203     .inorder = TRUE,
204     .final = function(x)
205       setNames(x, names(histone_data_request_ids))) %do%
206     {
207       deepblue_download_request_data(rid)
208     }
209
210   #map tiling regions to genes.
211   #Do this for one pair only, indices will be the same for other instances.
212   map_tiles_to_genes <- GenomicRanges::findOverlaps(
213     makeGRangesFromDataFrame(histone_tile_data[[1]]),
214     promoters(gene_expr_data[[1]],
215       downstream = 2500, upstream = 2500))
216
217   foreach(sample_pair = iter(metadata, by = "row")) %do%
218   {
219     gene_expr_sample <- gene_expr_data[[as.character(sample_pair$exprs_sample)]]
220     histone_sample <- histone_tile_data[[as.character(sample_pair$experiment_name)]]
221
222     combined <- cbind(histone_sample[as.data.frame(map_tiles_to_genes)$queryHits,],
223       as.data.frame(gene_expr_sample)[as.data.frame(map_tiles_to_genes)$
224         subjectHits,])
225
226     spread_combined <- combined %>%
227       group_by(`X.GENE_NAME.gencode.v22.`) %>%
228       filter(n() >= 49, n() <= 51) %>%
229       mutate(tile_number = row_number()) %>%
230       select(gene = `X.GENE_NAME.gencode.v22.`, TPM, mean_signal = `@AGG.MEAN`, tile_number)
231       %>%
232       spread(key = tile_number, value = mean_signal, convert = TRUE, fill = 0)
233
234     return(spread_combined)
235   }
236 }

```

```

235
236 names(histone_data) <- histone_marks
237 save.image(file = "matched_gene_expression_and_histone_data.RData")

```

Listing A.8: Predicting gene expression from histone data: loading the data from the *DeepBlue Server*. The source code is well annotated providing the details in each step.

```

1  library(glmnet)
2  library(tidyr)
3  library(ggplot2)
4  library(pROC)
5
6  #elastic net binomial regression with cross validation
7  models <- foreach(hmark = histone_marks, .inorder = TRUE) %do% {
8    hdata <- rbindlist(histone_data[[hmark]])
9    x <- log2(as.matrix(hdata[,3:53]) + 1e-6)
10   y <- log2(as.numeric(hdata$TPM) + 1e-6)
11   discrete_y <- rep(0, length(y))
12   discrete_y[which(y > median(y))] <- 1
13   discrete_y <- as.factor(discrete_y)
14
15   elnet_model <- cv.glmnet(x = x, y = discrete_y,
16                           family = "binomial", alpha = 0.5, type.measure = "auc")
17   elnet_roc <- roc(response = discrete_y,
18                  predictor = predict(elnet_model, newx = x, type = "response"),[1])
19   return(list(elnet_model, elnet_roc))
20 }
21
22 #plot ROC curves
23 plot(models[[1]][[2]], col = 1, xlim = c(1,0), ylim = c(0,1))
24 roc_legend <- paste(histone_marks[1], "(AUC = ",round(models[[1]][[2]]$auc,3),")",sep="")
25 for(i in 2:length(models)){
26   lines(models[[i]][[2]], col = i)
27   roc_legend <- c(roc_legend, paste(histone_marks[i], "(AUC = ",round(models[[i]][[2]]$auc,3),
28                                   ,")",sep=""))
29 }
30 legend("bottomright", roc_legend, lty = 1, col = seq_len(length(histone_marks)), inset =
31       0.05)
32
33 #plot coefficients for each tile and histone marks as heatmap
34 names(models) <- histone_marks
35 coefficients <- rbindlist(lapply(models, function(x){ as.data.frame(t(as.matrix(coef(x[[1]]))
36   )), idcol = "histone mark")
37 plot_data <- tidyr::gather(coefficients, key = "tile_id", value = "coefficient", 3:51)
38 plot_data$tile_id <- as.integer(plot_data$tile_id)
39 ggplot(plot_data, aes(x = tile_id, y = `histone mark`, fill = coefficient)) +
40   geom_tile() +
41   theme_bw() +
42   scale_fill_gradient2(low = "purple", high = "orange", mid = "white") +
43   geom_vline(aes(xintercept = 25.5)) +
44   geom_text(aes(x = 24, y = 1, label = "TSS")) +
45   labs(x = "tile number (100 bp each)")

```

Listing A.9: Predicting gene expression from histone data: building the models, executing the predictions, and displaying the results. The source code is well annotated providing the details in each step.

Constructing cell signatures

This use case aims to generate signatures of different BioSources for biomarkers identification. Due to its complexity, this use case is divided into five files:

- Listing A.10 is the main that loads the data, execute the functions provided by the other files, and show results in the form of heatmaps.
- Listing A.11 provides functions for retrieving the score matrices containing summarized data, as well for merging the data from the same BioSource providing information such as the score matrix mean for and standard deviation.
- Listing A.12 provide functions for computing the BioSource score and for generating their signatures.
- Listing A.13 contains the functions for plotting the heatmap, used for visual inspection.
- Listing A.14 provides a set of tests that verify if the filtering and ranking methods are correctly implemented.

```

1 # Load all package dependencies. install if missing!
2 library(DeepBlueR)
3 library(ggplot2)
4 library(dplyr)
5 library(foreach)
6 library(xlsx)
7 library(matrixStats)
8 library(stringr)
9 library(data.table)
10 library(gplots)
11 library(RColorBrewer)
12 library(doParallel)
13 library(testthat)
14
15 #register parallel processing, enable for speeding up signatures computation
16 #num_of_cores <- 37 #adjust as needed
17 cl <- parallel::makeCluster(4, outfile = "")
18 registerDoParallel(cl)
19
20 # Goal: establish a comprehensive list of marker regions that epigenetically identify a cell
    type of interest (e.g. for the development of cell type specific biomarkers)
21
22 ### Use Case 4: Obtaining region-specific DNA methylation data for a custom selection of cell
    types ###
23
24 # 1. Export a summary table that lists metadata for all samples that have sufficient DNA
    methylation data to be included in the analysis
25 # 2. The user manually adds a new column "custom_cell_type" to this table (in Excel),
    defining which biologically related samples should be aggregated into which cell types
    ("NA" for cell samples that are to be ignored)
26 # 3. The annotation table is imported into DeepBlue, and the use case continues with step 2
    of Use Case 1.
27
28 # first identify suitable biosource terms
29 biosource_blood <- deepblue_get_biosource_related("hematopoietic cell")
30
31 # Identify experimental files in BLUEPRINT for those biosource terms
32 selected_experiments <- deepblue_list_experiments(genome = "GRCh38",
33                                                    biosource = deepblue_extract_names(
34                                                        biosource_blood),
35                                                    project = "BLUEPRINT Epigenome",
36                                                    epigenetic_mark = "DNA methylation")
37 # keep only 'call' files

```

```

38 selected_experiments <- selected_experiments[grepl(pattern = "call\\.", selected_experiments$
    name),]
39
40 # keep only CpG methylation calls
41 selected_experiments <- selected_experiments[grepl(pattern = "CPG_methylation", selected_
    experiments$name),]
42
43 # download meta data
44 experiments_meta <- deepblue_meta_data_to_table(deepblue_extract_ids(selected_experiments))
45 experiments_meta$user_celltype <- experiments_meta$biosource_name
46
47 # write to XLSX
48 write.xlsx(experiments_meta, file = "experiments_metadata.xlsx")
49
50 # wait for user to edit the file and add custom cell type names
51 readline("Press enter when you are finished editing the metadata file experiments_metadata.
    xlsx")
52
53 # read edited meta data
54 experiments_meta <- read.xlsx(file = "experiments_metadata.xlsx", sheetIndex = 1)
55
56 # plot cell type sample numbers
57 plot_data <- experiments_meta %>% group_by(user_celltype) %>% summarize(count = n())
58
59 ggplot(plot_data, aes(x = user_celltype, y = count)) + geom_bar(stat = "identity") +
60   theme_bw() +
61   theme(axis.text.x = element_text(angle = 90, hjust = 1))
62
63
64 ### Use Case 1: Obtaining region-specific DNA methylation data for blood cell types ###
65 # 1. Within cell_type_class=blood, group all samples by cell_type
66 # 2. Select a region set of interest (e.g. a 1kb tiling of the genome or the Ensembl
    segmentation map)
67 # 3. Annotate each region with the mean and standard deviation (alternatively: median, 5th
    percentile, and 95th percentile) of the region's DNA methylation status in all samples
    of a given cell_type
68 # 4. Export the resulting [region] x [cell_type] table for manual filtering in R
69
70 # NOTE: not sure what you mean by grouping by cell type. grouping is not supported in
    DeepBlue and has to be done
71 # after downloading the data in R. please elaborate how it should group multiple samples, e.g
    . mean of the median and SD?
72
73 # Split the matrix generation by chromosome to make it more efficient.
74 # First it asks DeepBlue for the chromosome names it uses.
75 # NOTE: could be you also would want to remove the sex chromosomes here but it left them in
76
77 source("usecase1.R")
78
79 dna_meth_request_ids <- usecase_1_request_score_matrices(selected_experiments = selected_
    experiments,
80                                                         column_name = "VALUE",
81                                                         reference_genome = "GRCh38",
82                                                         experiments_meta = experiments_meta)
83
84 check_status(dna_meth_request_ids)
85
86 dna_meth_data <- usecase_1_download_score_matrices(request_ids = dna_meth_request_ids,
87                                                         experiments_meta = experiments_meta)
88
89 ### Use Case 2: Automated selection of cell-type specific biomarkers (this could also be done
    in R) ###
90
91 # 1. Based on the table produced in Use Case 1, add for each cell_type the following
    calculated measures of cell-type specificity:
92 # - number of cell types in which the region's mean DNA methylation is lower than the region'
    s DNA methylation in the selected cell_type

```

```

93 # - number of cell types in which the region's mean DNA methylation minus 1x the standard
    deviation is lower than the region's mean DNA methylation in the selected cell_type
94 # - number of cell types in which the region's mean DNA methylation minus 2x the standard
    deviation is lower than the region's mean DNA methylation in the selected cell_type
95 # 2. Rank the regions based on the worst rank of the three metrics (each one ranked
    individually and then taking the row-wise maximum)
96 # 3. Return the top-500 regions that are hypomethylated (lower methylation) in the selected
    cell_type based on the consensus ranking
97 # 4. Repeat steps 1 to 3 with a focus on hypermethylated regions (higher methylation) in the
    selected cell_type
98
99 source("usecase2.R")
100
101 hypo_meth_ranks <- compute_cell_type_scores(dna_meth_data$mean_matrix,
102                                           dna_meth_data$sd_matrix)
103
104 hyper_meth_ranks <- compute_cell_type_scores(dna_meth_data$mean_matrix,
105                                           dna_meth_data$sd_matrix,
106                                           invert = TRUE)
107
108 #check if the rank computations are correct using a random region and cell type
109 source("testing.R")
110 testing(dna_meth_data$mean_matrix, dna_meth_data$sd_matrix, hypo_meth_ranks, hyper_meth_ranks
111        )
112 #get unique biosources for extracting signatures
113 unique_biosources <- as.character(unique(experiments_meta$user_celltype))
114
115 #min.num.of.regions -> include at least 500 regions, include remaining regions with the same
    rank score
116 #max.num.of.regions -> if more than x regions have been selected draw a random subset of x
117 #choose small x for example to make heatmaps feasible.
118 hyper_signatures <- generate_cell_type_signatures(unique_biosources, dna_meth_data$regions,
119                                                  hyper_meth_ranks$`worst rank`,
120                                                  min.num.of.regions = 100,
121                                                  max.num.of.regions = NULL)
122 hypo_signatures <- generate_cell_type_signatures(unique_biosources, dna_meth_data$regions,
123                                                  hypo_meth_ranks$`worst rank`,
124                                                  min.num.of.regions = 100,
125                                                  max.num.of.regions = NULL)
126
127 #plot size of individual signatures and heatmaps of a subset of the signatures
128
129 # plot number of signature regions for each cell type.
130
131 hyper_signatures_df <- rbindlist(hyper_signatures, idcol = "cell_type")
132 hyper_signatures_df$status <- "hyper"
133
134 hypo_signatures_df <- rbindlist(hypo_signatures, idcol = "cell_type")
135 hypo_signatures_df$status <- "hypo"
136
137 ggplot(data = rbind(hypo_signatures_df, hyper_signatures_df),
138        aes(x = cell_type)) +
139   geom_bar(stat = "count") +
140   theme_bw() +
141   facet_wrap(~status, ncol = 1) +
142   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
143
144 # bonus: plot a heatmap with smaller signatures
145
146 hyper_signatures_heatmap <- generate_cell_type_signatures(unique_biosources, dna_meth_data$
    regions,
147                                                  hyper_meth_ranks$`worst rank`,
148                                                  min.num.of.regions = 100,
149                                                  max.num.of.regions = 100)
150 hypo_signatures_heatmap <- generate_cell_type_signatures(unique_biosources, dna_meth_data$
    regions,

```

```

151                                     hypo_meth_ranks$`worst rank`,
152                                     min.num.of.regions = 100,
153                                     max.num.of.regions = 100)
154
155
156 source("heatmap.R")
157 plot_heatmap(signatures = hypo_signatures_heatmap,
158             mean_df_matrix = dna_meth_data$mean_df,
159             experiments_meta = experiments_meta,
160             signature_score_numeric = signature_score_numeric,
161             filename = "DNA_meth_hypo")
162
163 plot_heatmap(signatures = hyper_signatures_heatmap,
164             mean_df_matrix = dna_meth_data$mean_df,
165             experiments_meta = experiments_meta,
166             signature_score_numeric = signature_score_numeric,
167             filename = "DNA_meth_hyper")
168
169
170 #cleanup
171 if(exists("cl")) stopCluster(cl)

```

Listing A.10: Generating biosource signatures for biomarkers identification: the main file loads the data, execute the functions provided by the other files, and show results in the form of heatmaps.

```

1 usecase_1_request_score_matrices <- function(selected_experiments, column_name,
2                                             reference_genome = "GRCh38",
3                                             experiments_meta)
4 {
5   chromosomes <- grep("_", deepblue_extract_ids(deepblue_chromosomes(reference_genome)),
6         invert = TRUE, value = TRUE)
7
8   # It also need to define the name of the column it need in the WIG files
9   experiments_columns <- deepblue_select_column(selected_experiments, column_name)
10
11   request_ids <- foreach(chromosome = chromosomes,
12     .final = function(x) setNames(x, chromosomes)) %do% {
13
14     #Select BLUEPRINT Ensembl regulatory build regions
15     #deepblue_list_annotations(genome = reference_genome) for all
16     annotations
17     ensembl_reg_build <- deepblue_select_annotations(annotation_name =
18       "Blueprint Ensembl Regulatory Build",
19       genome =
20       reference_
21       genome,
22       chromosome =
23       chromosome)
24
25     #alternatively if you want to use 1kb tiling
26     #deepblue_tiling_regions(size = 1000, genome = reference_genome,
27       chromosome = chromosome)
28
29     # request the score matrices
30     experiments_mean_request_id <-
31       deepblue_score_matrix(experiments_columns = experiments_columns,
32         aggregation_function = "mean",
33         aggregation_regions_id = ensembl_reg_build
34       )
35
36     experiments_sd_request_id <-
37       deepblue_score_matrix(experiments_columns = experiments_columns,
38         aggregation_function = "sd",
39         aggregation_regions_id = ensembl_reg_build
40       )

```

```

32
33         #collect all request ids in pairs
34         list(experiments_mean_request_id, experiments_sd_request_id)
35     }
36     return(request_ids)
37 }
38
39 check_status <- function(request_ids){
40     table(unlist(lapply(deepblue_info(unlist(request_ids)), function(x) x$state)))
41 }
42
43 usecase_1_download_score_matrices <- function(request_ids,
44                                             experiments_meta)
45 {
46     #check status of all requests
47     state <- check_status(request_ids)
48
49     while(names(state) != c("done")){
50         print(state)
51         message("At least one request is not completed. Waiting...")
52         Sys.sleep(10)
53     }
54
55     # once finished, download the score matrices and merge them
56     download_and_combine_score_matrices <- function(request_ids, i){
57         # NOTE bind_rows will match columns by name such that it don't have to worry about column
58         # order
59         foreach(pair_of_request_ids = request_ids, .combine = bind_rows) %do%
60         {
61             request_id <- pair_of_request_ids[[i]]
62             if(deepblue_info(request_id)$state != "done")
63                 stop(paste(request_id, "is not reported as done. Please check its (error) state.))
64             deepblue_download_request_data(request_id)
65         }
66     }
67     experiments_mean_score_matrix <- download_and_combine_score_matrices(request_ids, 1)
68     experiments_sd_score_matrix <- download_and_combine_score_matrices(request_ids, 2)
69
70     # extract regions coordinates
71     regions <- experiments_mean_score_matrix[,1:3]
72
73     # convert the score matrix to a numeric matrix (omitting chr, start, end columns)
74     experiments_mean_score_numeric <- as.matrix(experiments_mean_score_matrix[,4:ncol(
75         experiments_mean_score_matrix)])
76     experiments_sd_score_numeric <- as.matrix(experiments_sd_score_matrix[,4:ncol(experiments_
77         sd_score_matrix)])
78
79     # the column order is not guaranteed to be identical in these two matrices, thus it
80     # reorders the second one
81     experiments_sd_score_numeric <- experiments_sd_score_numeric[,colnames(experiments_mean_
82         score_numeric)]
83
84     # group by cell type
85     unique_biosources <- unique(experiments_meta$user_celltype)
86
87     # multiple columns in the matrices correspond to the same cell type. here it merges them by
88     # mean, omitting NAs.
89     aggregate_by_cell_type <- function(score_matrix){
90         foreach(biosource = unique_biosources,
91             .combine = cbind,
92             .final = function(x) {
93                 colnames(x) <- unique_biosources;
94                 return(x)}) %do% {
95             experiments_with_current_biosource <- experiments_meta[which(experiments_meta
96                 $user_celltype == biosource), "name"]
97             subset_of_matrix <- score_matrix[,experiments_with_current_biosource]
98             if(is.null(ncol(subset_of_matrix))) return(subset_of_matrix)
99         }
100     }

```

Listing A.11: Generating biosource signatures for biomarkers identification: this file retrieves the data in the form of score matrices, containing summarized data, as well for merging the data from the same BioSource providing information such as the score matrix mean for and standard deviation

```

1 #hypomethylated regions
2
3 compute_cell_type_scores <- function(mean_score_matrix, sd_score_matrix, invert = FALSE){
4
5   if(invert){
6     sign_for_score <- -1
7     direction <- "higher"
8     sign_for_label <- "+"
9   }
10  else{
11    sign_for_score <- 1
12    direction <- "lower"
13    sign_for_label <- "-"
14  }
15  #the three metrices described above. The first one is simply the rank
16  mean_score <- t(apply(sign_for_score * mean_score_matrix,
17                        1, rank, ties.method = "max") - 1)
18
19  #the other two are more complicated since they need to get the rank of the average
20  #DNA methylation in the matrix that was modified by subtracting the SD...
21  #It thus adds the average DNA methylation to the SD modified matrix and rank them together
22  #This has to be done for each cell type individually.
23
24  compute_ranks_in_sd_matrix <- function(multiplier){
25
26    overall_result <- foreach(cell_type_index = seq_len(ncol(mean_score_matrix)),
27                              .export = c("mean_score_matrix", "sd_score_matrix", "sign_for_score"),
28                              .combine = cbind,
29                              .inorder = TRUE, #otherwise the cell types would be mixed up in paralellization
30                              .final = function(x) {

```

```

31         colnames(x) <- colnames(mean_score_matrix)
32         rownames(x) <- rownames(mean_score_matrix)
33         return(x)
34     }) %dopar%
35     {
36         current_cell_type <- mean_score_matrix[,cell_type_index]
37         result <- t(apply(sign_for_score * cbind(current_cell_type,
38             (mean_score_matrix[, -cell_type_index] - multiplier * sd_score_matrix[, -cell_type_
39                 index])),
40                 1, rank, ties.method = "max") - 1)
41         return(result[,1]) #return only first column with ranks of the respective cell type
42     }
43     return(overall_result)
44 }
45
46 mean_1_sd <- compute_ranks_in_sd_matrix(1)
47 mean_2_sd <- compute_ranks_in_sd_matrix(2)
48
49 #the worst rank
50 worst_rank <- pmax(mean_score, mean_1_sd, mean_2_sd)
51
52 #combine regions with ranks
53 result <- list(mean_score, mean_1_sd, mean_2_sd, worst_rank)
54 names(result) <- c(paste("number of cell types with score", direction, "than average"),
55     paste("number of cell types with score", direction, "than (average",
56         sign_for_label, "SD)"),
57     paste("number of cell types with score", direction, "than (average",
58         sign_for_label, "2*SD)"),
59     "worst rank")
60
61 return(result)
62 }
63
64 #generate a list of cell type signatures
65 generate_cell_type_signatures <- function(unique_biosources, regions, ranks,
66     min.num.of.regions = 500,
67     max.num.of.regions = NULL){
68
69     if(!is.null(max.num.of.regions))
70         if(min.num.of.regions > max.num.of.regions)
71             stop("max.num.of.regions needs to be equal to or larger than min.num.of.regions.")
72
73     foreach(biosource = unique_biosources,
74         .final = function(x) setNames(x, unique_biosources)) %do% {
75         regions_ranks <- cbind(regions, ranks)
76         result <- regions_ranks %>%
77             select(CHROMOSOME, START, END, celltypes_scoring_better = UQ(biosource)) %>%
78             top_n(min.num.of.regions, -celltypes_scoring_better)
79         if(!is.null(max.num.of.regions)){
80             #return a random sample of regions
81             result <- result %>% sample_n(min(nrow(result), max.num.of.regions))
82         }
83         return(result)
84     }
85 }

```

Listing A.12: Generating biosource signatures for biomarkers identification: this file provides the functions for computing the BioSource scores and generating their signatures.

```

1 plot_heatmap <- function(signatures, mean_df_matrix, experiments_meta,
2     signature_score_numeric, filename)
3 {
4     signature_regions <- rbindlist(signatures) %>% select(CHROMOSOME, START, END) %>% distinct
5     ()
6

```



```

7  # filter the DNA methylation matrix for those signature regions
8  signature_score_matrix <- dplyr::semi_join(mean_df_matrix,
9                                           signature_regions,
10                                          by = c("CHROMOSOME", "START", "END"))
11  signature_score_numeric <- as.matrix(signature_score_matrix[,4:ncol(signature_score_matrix)
12                                     ])
13  signature_score_numeric[is.na(signature_score_numeric)] <- 0
14  rownames(experiments_meta) <- experiments_meta$name
15  unique_biosources <- unique(experiments_meta$user_celltype)
16
17  # define cell types and colors
18  getPalette <- colorRampPalette(brewer.pal(9, "Set1"))
19
20  color_map <- data.frame(biosource = unique_biosources,
21                        color = getPalette(length(unique_biosources)))
22
23  exp_names <- colnames(signature_score_numeric)
24  biosource_colors <- data.frame(name = exp_names, biosource = experiments_meta[exp_names, "
25                                user_celltype"])
26  biosource_colors <- left_join(biosource_colors, color_map, by = "biosource")
27  color_vector <- as.character(biosource_colors$color)
28  names(color_vector) <- biosource_colors$biosource
29
30  #plot heatmap using pearson correlation for hierarchical clustering
31  pdf(file = paste0(filename, "_heatmap.pdf"), paper = "a4")
32  heatmap.2(signature_score_numeric, labRow = NA, labCol = NA,
33            trace = "none", ColSideColors = color_vector,
34            hclust=function(x) hclust(x,method="complete"),
35            distfun=function(x) as.dist(1-cor(t(x), method = "pearson")), Rowv = TRUE,
36            dendrogram = "column",
37            key.xlab = "beta value", denscol = "black", keysize = 1.5,
38            key.par = list(mar = c(8.5, 2.5, 1, 1)), key.title = NA)
39
40  # Next, it adds a legend showing which cell type has which color
41  plot.new()
42  legend(x = 0, y = 1,
43        legend = color_map$biosource,
44        col = as.character(color_map$color),
45        text.width = 0.6,
46        lty= 1,
47        lwd = 6,
48        cex = 0.7,
49        y.intersp = 0.7,
50        x.intersp = 0.7,
51        inset=c(-0.21,-0.11))
52  dev.off()
53 }

```

Listing A.13: Generating biosource signatures for biomarkers identification: this file provides functions for displaying results in the form of heatmaps.

```

1  testing <- function(mean_matrix, sd_matrix, hypo_meth_ranks, hyper_meth_ranks){
2    mean_test <- mean_matrix[12345,]
3    sd_test <- sd_matrix[12345,]
4    mean_1_sd_test <- mean_test - sd_test
5    mean_2_sd_test <- mean_test - 2 * sd_test
6
7    #test hypo methylation results
8    testthat::expect_equal(hypo_meth_ranks$`number of cell types with score lower than average
9                          `[12345,],
10                           (rank(mean_test, ties.method = "max" )-1))
11
12    testthat::expect_equal(hypo_meth_ranks$`number of cell types with score lower than (average
13                          - SD)`[12345,5],

```

```

12         (rank(c(mean_test[5], mean_1_sd_test[-5]), ties.method = "max" )-1)
13         [1])
14     testthat::expect_equal(hypo_meth_ranks$`number of cell types with score lower than (average
15         - 2*SD)`[12345,35],
16         (rank(c(mean_test[35], mean_2_sd_test[-35]), ties.method = "max" )
17         -1)[1])
18     #test hyper methylation results
19     testthat::expect_equal(hyper_meth_ranks$`number of cell types with score higher than
20         average`[12345,],
21         (rank(-mean_test, ties.method = "max" )-1))
22     testthat::expect_equal(hyper_meth_ranks$`number of cell types with score higher than (
23         average + SD)`[12345,5],
24         (rank(c(-mean_test[5], -mean_1_sd_test[-5]), ties.method = "max" )
25         -1)[1])
26     testthat::expect_equal(hyper_meth_ranks$`number of cell types with score higher than (
27         average + 2*SD)`[12345,35],
28         (rank(c(-mean_test[35], -mean_2_sd_test[-35]), ties.method = "max" )
29         -1)[1])
30 }

```

Listing A.14: Generating biosource signatures for biomarkers identification: this file tests the executed filtering and calculation method, ensuring a better method and code quality.

A.3.3 DIVE

DeepBlue and DIVE integration example: Comparing data from DEEP and CREST

One of *DIVE* limitations is the lack of out of the box support for signal files. It is possible to overcome this limitation by developing a script for processing the signal files using the *DeepBlue Server* API. Listing A.15 shows how to aggregate and filter DNA methylation data for analyzing in *DIVE*. This source code uses two experiments from *DEEP* and two from *CREST*, filtering each one for hypomethylated and hypermethylated regions, generating in the total 8 *query IDs* that can be easily loaded into *DIVE* for further analysis.

```

1  import xmlrpclib
2
3  url = "http://deepblue.mpi-inf.mpg.de/xmlrpc"
4  deepblue = xmlrpclib.Server(url, allow_none=True)
5
6  USER_KEY = "anonymous_key"
7
8  GENOME = "hg19"
9  GENE_MODEL = "genecode v19"
10
11 def get_promoters(gene_model):
12     (_, q_genes) = deepblue.select_genes(
13         None, None, gene_model, None, None, None, USER_KEY)
14     (_, promoters) = deepblue.flank(q_genes, -2500, 2000, True, USER_KEY)
15     return promoters
16
17 def get_data_id(project, genome):
18     # Obtain the promoters
19     promoters = get_promoters(GENE_MODEL)
20
21     # List all mRNA experiments from Hepatocytes and from the CREST project.
22     _, experiments = deepblue.list_experiments(genome, "signal",
23         "DNA Methylation", "hepatocyte",

```

```

24                                     None, None, project,
25                                     USER_KEY)
26     # Remove the coverage files
27     selected_exps = [e_name for _, e_name]
28         in experiments if "coverage" not in e_name.lower()]
29
30     # Use only two experiments of each project
31     for selected_exp in selected_exps[0:2]:
32         _, e_query = deepblue.select_experiments(
33             selected_exp, None, None, None, USER_KEY)
34
35         # Aggregate the regions using the column VALUE
36         (_, agg_id) = deepblue.aggregate(
37             e_query, promoters, "VALUE", USER_KEY)
38
39         _, low_id = deepblue.filter_regions(
40             agg_id, "@AGG.MEAN", "<=", "15", "number", USER_KEY)
41         _, high_id = deepblue.filter_regions(
42             agg_id, "@AGG.MEAN", ">=", "85", "number", USER_KEY)
43
44         # Just for speed up the data loading in DIVE
45         deepblue.count_regions(low_id, USER_KEY)
46         deepblue.count_regions(high_id, USER_KEY)
47
48         print selected_exp, low_id, high_id
49
50 get_data_id("DEEP (IHEC)", GENOME)
51 get_data_id("CREST", GENOME)

```

Listing A.15: Pre-processing data for comparing data from *DEEP* and *CREST* in *DIVE*: this source code is straightforward, where in lines 1 - 9 load the required libraries and set the parameters used in the data processing. Lines 11-15 provides a function that return a *query ID* which references promoter regions. The function *get_data* in lines 17 - 48 obtain the promoters, than the list of experiments, which is filtered for removing the DNA methylation coverage files, two of the remaining files are individually aggregated by the promoter regions, and the aggregation results are filtered for hypomethylated (*DNAmethylationlevel* ≤ 15) and hypermethylated (*DNAmethylationlevel* ≥ 85) regions. The filtering *query IDs* are displayed to the user, which must load them in *DIVE*. Lines 50-51 execute the *get_data* function for the public *DEEP* and *CREST* data.

A.4 DeepBlue Server API

The *DeepBlue Server* API has 96 operations. These operations are divided into 19 categories: Status, Column Types, Genomic Regions Operations, Requests, BioSources relationship, Epigenetic marks, Utilities, Genes, Genomes, Data Modification, General Information, BioSources, Experiments, Samples, Genomic Regions Enrichment, Expressions, Annotations, Projects, Techniques.

The API operations documentation can be programmatically retrieved using the *commands()* operation. As an example, the following operations list was generated by a python script.

In the following, each category is presented with with its respective operations.

Annotations

Inserting and listing annotations

add_annotation(*name, genome, description, data, format, extra_metadata, user_key*)

Add a custom annotation of genomic regions such as, for instance, promoters, transcription factor binding sites, or genes to DeepBlue. Annotations are a set genomic regions such as, for instance, promoters, transcription factor binding sites, or genes to DeepBlue.

find_motif(*motif, genome, chromosomes, start, end, overlap, user_key*)

Find genomic regions based on a given motif that appears in the genomic sequence.

list_annotations(*genome, user_key*)

List all annotations of genomic regions currently available in DeepBlue.

BioSources

Inserting and listing biosources

add_biosource(*name, description, extra_metadata, user_key*)

Add a BioSource to DeepBlue. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

list_biosources(*extra_metadata, user_key*)

List BioSources included in DeepBlue. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line. It is possible to filter the BioSources by their *extra_metadata* fields content. These fields vary depending on the original data source.

list_similar_biosources(*name, user_key*)

List all BioSources that have a similar name compared to the provided name. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line. The similarity is calculated using the Levenshtein method.

BioSources relationship

Set the relationship between different biosources

create_experiments_set(*name, description, public, experiment_name, user_key*)

Create a set of experiments to be shared among other users

get_biosource_children(*biosource, user_key*)

A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line. These form a hierarchy in which children of a BioSource term can be fetched with this command. Children terms are more specific terms that are defined in the imported ontologies.

get_biosource_parents(*biosource, user_key*)

A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line. These form a hierarchy in which the parent of a BioSource term can be fetched with this command. Parent terms are more generic terms that are defined in the imported ontologies.

get_biosource_related(*biosource, user_key*)

A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line. These form a hierarchy in which the children of a BioSource term and its synonyms can be fetched with this command. Children terms are more specific terms that are defined in the imported ontologies. Synonyms are different aliases for the same biosource.

get_biosource_synonyms(*biosource, user_key*)

Obtain the synonyms of the specified biosource. Synonyms are different aliases for the same biosource. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

set_biosource_parent(*parent, child, user_key*)

Define a BioSource as parent of another BioSource. This command is used to build the BioSources hierarchy. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

set_biosource_synonym(*biosource, synonym_name, user_key*)

Define a synonym for a BioSource. BioSources can have multiple synonyms. This command for can be used multiply to add several synonyms. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

Column Types

Inserting and listing different column types

create_column_type_calculated(*name, description, code, user_key*)

Create a calculated column type in DeepBlue. A calculated column can use existing columns and transform or summarize them through mathematical operations or string operations using the programming language LUA. Examples: the following 'code' parameter can be used to calculate the square root of the column VALUE: 'return math.sqrt(value_of('VALUE'))'. Another example is dividing the value of the column 'VALUE' by the region length: 'return value_of('VALUE') / (value_of('END') - value_of('START'))'.

create_column_type_category(*name, description, items, user_key*)

Create a categoric column type in DeepBlue from a set of items. As example, the STRAND column is a category column that contain the items: '+', '-', and '.'.

create_column_type_range(*name, description, minimum, maximum, user_key*)

Create a range column type in DeepBlue. For example, a METHYLATION_BETA_VALUE column where accepted values are from 0.0 to 1.0.

create_column_type_simple(*name, description, type, user_key*)

Create a simple column type (string, integer, double) in DeepBlue.

list_column_types(*user_key*)

Lists the ColumnTypes included in DeepBlue.

Data Modification

Operations that modify the data content

change_extra_metadata(*id, key, value, user_key*)

Modify the *extra/metadata* content of experiments, annotations, biosources, and samples. Use this command with an *extra-metadata* key without value for removing this key. *Extra-metadata* fields are optional non-standardized fields that are created during the import process. Only files uploaded by the user can be modified. The command 'clone_dataset' must be used if the user wants to modify a files that does not belong to him.

clone_dataset(*dataset_id, new_name, new_epigenetic_mark, new_sample, new_technique, new_project, description, format, extra_metadata, user_key*)

Clone a dataset optionally changing its metadata and extra_metadata values. This command must be used in data curation because users do not have permission to change the metadata values of the Annotations and Experiments that were not uploaded by them.

Epigenetic marks

Inserting and listing epigenetic marks

add_epigenetic_mark(*name, description, extra_metadata, user_key*)

Include an Epigenetic Mark such as, for instance, a specific type of histone modification, in DeepBlue.

list_epigenetic_marks(*extra_metadata, user_key*)

List Epigenetic Marks included in DeepBlue. This includes histone marks, DNA methylation, DNA sensitivity, etc. It is possible to filter the Epigenetic Marks by their *extra_metadata* field content.

list_similar_epigenetic_marks(*name, user_key*)

List all Epigenetic Marks that have a similar name compared to the provided name. The similarity is calculated using the Levenshtein method.

Experiments

Inserting and listing experiments

add_experiment(*name, genome, epigenetic_mark, sample, technique, project, description, data, format, extra_metadata, user_key*)

Add an Experiment in DeepBlue. An Experiment describes the characteristics of a specific Epigenetic Mark with respect to a single sample. The technology used and project must be informed as well. *Extra-metadata* can be specified in addition to the mandatory meta information.

collection_experiments_count(*controlled_vocabulary, genome, type, epigenetic_mark, biosource, sample, technique, project, user_key*)

Count the number of experiments that match the selection criteria in each term of the selected *controlled_vocabulary*. The selection can be achieved through specifying a list of BioSources, experimental Techniques, Epigenetic Marks, Samples or Projects.

faceting_experiments(*genome, type, epigenetic_mark, biosource, sample, technique, project, user_key*)

Summarize the *controlled_vocabulary* fields, from experiments that match the selection criteria. It is similar to the 'collection_experiments_count' command, but this command return the summarization for all *controlled_vocabulary* terms.

list_experiments(*genome, type, epigenetic_mark, biosource, sample, technique, project, user_key*)

List the DeepBlue Experiments that matches the search criteria defined by this command parameters.

list_recent_experiments(*days, genome, epigenetic_mark, sample, technique, project, user_key*)

List the latest Experiments included in DeepBlue that match criteria defined in the parameters. The returned experiments are sorted by insertion date.

list_similar_experiments(*name, genome, user_key*)

List all Experiments that have a similar name compared to the provided name. The similarity is calculated using the Levenshtein method.

preview_experiment(*experiment_name, user_key*)

List the DeepBlue Experiments that matches the search criteria defined by this command parameters.

Expressions

Expression data

add_expression(*expression_type, sample_id, replica, data, format, project, extra_metadata, user_key*)

Include Expression data in DeepBlue.

list_expressions(*expression_type, sample_id, replica, project, user_key*)

List the Expression currently available in DeepBlue. An expression is a set of data with an identifier and an expression value.

select_expressions(*expression_type, sample_ids, replicas, identifiers, projects, gene_model, user_key*)

Select expressions (by their name or ID) as genomic regions from the specified model.

General Information

Commands for all types of data

cancel_request(*id, user_key*)

Stop, cancel, and remove request data. The request processed data is remove if its processing was finished.

info(*id, user_key*)

Information about a DeepBlue data identifier (ID). Any DeepBlue data ID can be queried with this command. For example, it is possible to obtain all available information about an Experiment using its ID, to obtain the actual Request processing status or the information about a Sample. A user can obtain information about him- or herself using the value 'me' in the parameter 'id'. Multiple IDs can be queried in the same operation.

is_biosource(*biosource, user_key*)

Verify if the name is an existing and valid DeepBlue BioSource name. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

list_in_use(*controlled_vocabulary, user_key*)

List all terms used by the Experiments mandatory metadata that have at least one Experiment or Annotation using them.

name_to_id(*name, collection, user_key*)

Obtain the data ID(s) from the informed data name(s).

remove(*id, user_key*)

Remove a DeepBlue data by using its ID.

search(*keyword, type, user_key*)

Search all data of all types for the given keyword. A minus (-) character in front of a keyword searches for data without the given keyword. The search can be restricted to the following data types are: Annotations, Biosources, Column_types, Epigenetic_marks, Experiments, Genomes, Gene_models, Gene_expressions, Genes, Gene_ontology, Projects, Samples, Techniques, Tilings.

Genes

Gene models and genes identifiers

add_gene_model(*gene_model, genome, description, data, format, extra_metadata, user_key*)

Include a Gene Model in DeepBlue. The data must be in the GTF format. Important: this command will include only lines where the column 'feature' is 'genes'.

add_gene_ontology_term(*go_id, go_label, description, namespace, user_key*)

Add a Gene Ontology Term to DeepBlue. A Gene Ontology Term refers to a term use to describe the genes functions.

annotate_gene(*gene_ensb_id, go_term_id, user_key*)

Annotate a Gene with a Gene Ontology Term.

count_gene_ontology_terms(*genes, go_terms, chromosome, start, end, gene_model, user_key*)

Summarize the controlled_vocabulary fields, from experiments that match the selection criteria. It is similar to the 'collection_experiments_count' command, but this command return the summarization for all controlled_vocabulary terms.

list_gene_models(*user_key*)

List all the Gene Models currently available in DeepBlue. A gene model is a set of genes usually imported from GENCODE. For example Gencode v22.

list_genes(*genes, go_terms, chromosome, start, end, gene_model, user_key*)

List the Genes currently available in DeepBlue.

select_genes(*genes, go_terms, gene_model, chromosome, start, end, user_key*)

Select genes (by their name or ID) as genomic regions from the specified gene model.

set_gene_ontology_term_parent(*parent_go_id, parent_go_id, user_key*)

Define a BioSource as parent of another BioSource. This command is used to build the BioSources hierarchy. A BioSource refers to a term describing the origin of a given sample, such as a tissue or cell line.

Genomes

Inserting and listing genomes

add_genome(*name, description, data, user_key*)

Add a (reference) Genome assembly to DeepBlue.

chromosomes(*genome, user_key*)

List the chromosomes of a given Genome.

list_genomes(*user_key*)

List Genomes assemblies that are registered in DeepBlue.

list_similar_genomes(*name, user_key*)

Lists all Genomes that have a similar name compared to the provided name. The similarity is calculated using the Levenshtein method.

upload_chromosome(*genome, chromosome, data, user_key*)

Upload the DNA sequence of a chromosome.

Genomic Regions Enrichment

Enrich the genome regions

enrich_regions_fast(*query_id, genome, epigenetic_mark, biosource, sample, technique, project, user_key*)

Enrich the regions based on regions bitmap signature comparison.

enrich_regions_go_terms(*query_id, gene_model, user_key*)

Enrich the regions based on Gene Ontology terms.

enrich_regions_overlap(*query_id, background_query_id, datasets, genome, user_key*)

Enrich the regions based on regions overlap analysis.

Genomic Regions Operations

Operating on the data regions

aggregate(*data_id, ranges_id, column, user_key*)

Summarize the *data_id* content using the regions specified in *ranges_id* as boundaries. Use the fields @AGG.MIN, @AGG.MAX, @AGG.SUM, @AGG.MEDIAN, @AGG.MEAN, @AGG.VAR, @AGG.SD, @AGG.COUNT in 'get_regions' command 'format' parameter to retrieve the computed values minimum, maximum, median, mean, variance, standard deviation and number of regions, respectively.

binning(*query_data_id, column, bins, user_key*)

Bin results according to counts.

count_regions(*query_id, user_key*)

Return the number of genomic regions present in the query.

coverage(*query_id, genome, user_key*)

Send a request to count the number of regions in the result of the given query.

distinct_column_values(*query_id, field, user_key*)

Obtain the distinct values of the field.

extend(*query_id, length, direction, use_strand, user_key*)

Extend the genomic regions included in the query. It is possible to extend downstream, upstream or in both directions.

filter_by_motif(*query_id, motif, user_key*)

Filter the genomic regions by a regular expression motif.

filter_regions(*query_id, field, operation, value, type, user_key*)

Filter the genomic regions by their content.

flank(*query_id, start, length, use_strand, user_key*)

Create a set of genomic regions that flank the query regions. The original regions are removed from the query. Use the merge command to combine flanking regions with the original query.

get_experiments_by_query(*query_id, user_key*)

List the experiments and annotations that have at least one genomic region in the final query result.

get_regions(*query_id, output_format, user_key*)

Trigger the processing of the query's genomic regions. The output is a column-based format with columns as defined in the 'output_format' parameter. Use the command 'info' for verifying the processing status. The 'get_request_data' command is used to download the regions using the programmatic interface. Alternatively, results can be download using the URL: http://deepblue.mpi-inf.mpg.de/download?r_id=<request_id>&key=<user_key>.

input_regions(*genome, region_set, user_key*)

Upload a set of genomic regions that can be accessed through a query ID. An interesting use case for this command is to upload a set of custom regions for intersecting with genomic regions in DeepBlue to specifically select regions of interest.

intersection(*query_data_id, query_filter_id, user_key*)

Select genomic regions that intersect with at least one region of the second query. This command is a simplified version of the 'overlap' command.

merge_queries(*query_a_id, query_b_id, user_key*)

Merge regions from two queries in a new query.

overlap(*query_data_id, query_filter_id, overlap, amount, amount_type, user_key*)

Select genomic regions that overlap or not overlap with with the specified number of regions of the second query. Important: This command is still experimental and changes may occur.

query_cache(*query_id, cache, user_key*)

Cache a query result in DeepBlue memory. This command is useful when the same query ID is used multiple times in different requests. The command is an advice for DeepBlue to cache the query result and there is no guarantee that this query data access will be faster.

query_experiment_type(*query_id, type, user_key*)

Filter the query ID for regions associated with experiments of a given type. For example, it is possible to select only peaks using this command with the 'peaks' parameter.

score_matrix(*experiments_columns, aggregation_function, aggregation_regions_id, user_key*)

Build a matrix containing the aggregation result of the experiments data by the aggregation boundaries.

select_annotations(*annotation_name, genome, chromosome, start, end, user_key*)

Select regions from the Annotations that match the selection criteria.

select_experiments(*experiment_name, chromosome, start, end, user_key*)

Selects regions from Experiments by the experiments names.

select_regions(*experiment_name, genome, epigenetic_mark, sample_id, technique, project, chromosomes, start, end, user_key*)

Selects Experiment regions that matches the criteria informed by the operation parameters.

tiling_regions(*size, genome, chromosome, user_key*)

Generate tiling regions across the genome chromosomes. The idea is to "bin" genomic regions systematically in order to obtain disjoint regions over which one can aggregate. Using the 'score_matrix' command, these bins (tiles) can be compared directly across experiments.

Projects

Inserting and listing projects

add_project(*name, description, user_key*)

Add a Project to DeepBlue. A Project is used to group Experiments and to define their origin.

add_user_to_project(*user, project, set, user_key*)

Add a user as Project member.

list_projects(*user_key*)

List Projects included in DeepBlue.

list_similar_projects(*name, user_key*)

List Projects that have a similar name compared to the provided name. The similarity is calculated using the Levenshtein method.

set_project_public(*project, set, user_key*)

Define a project as public. This means that all DeepBlue users can then access its data. You must be the project owner to perform this operation.

Requests

Requests status information and results

get_request_data(*request_id, user_key*)

Download the requested data. The output can be (i) a string (get_regions, score_matrix, and count_regions), or (ii) a list of ID and names (get_experiments_by_query), or (iii) a struct (coverage).

list_requests(*request_state, user_key*)

List the Requests made by the user. It is possible to obtain only the requests of a given state.

reprocess(*request_id, user_key*)

Reprocess the request. Useful when the request was cancelled or removed.

Samples

Inserting and listing samples

add_sample(*biosource, extra_metadata, user_key*)

Add a Sample to DeepBlue that is related to a BioSource.

add_sample_from_gsm(*biosource, gsm_id, user_key*)

Add a Sample to DeepBlue that is related to a BioSource and can be linked to an existing GSM identifier (from a GEO repository).

list_samples(*biosource, extra_metadata, user_key*)

List Samples included in DeepBlue. It is possible to filter by the BioSource and by extra_metadata fields content.

Status

Checking DeepBlue status

commands()

List all available DeepBlue commands.

echo(*user_key*)

Greet the user with the DeepBlue version.

Techniques

Inserting and listing techniques

add_technique(*name, description, extra_metadata, user_key*)

Add an experimental Technique to DeepBlue.

list_similar_techniques(*name, user_key*)

List Techniques that have a similar name compared to the provided name. The similarity is calculated using the Levenshtein method.

list_techniques(*user_key*)

List the Techniques included in DeepBlue.

Utilities

Utilities for connecting operations

extract_ids(*list*)

A utility command that returns a list of IDs extracted from a list of ID and names.

extract_names(*list*)

A utility command that returns a list of names extracted from a list of ID and names.

A.5 List of publications

Albrecht, F., List, M., Bock, C., and Lengauer, T. (2016). “DeepBlue epigenomic data server: programmatic data retrieval and analysis of epigenome region sets.” *Nucleic Acids Research*. DOI: [10.1093/nar/gkw211](https://doi.org/10.1093/nar/gkw211).

Albrecht, F., List, M., Bock, C., and Lengauer, T. (2017). “DeepBlueR: large-scale epigenomic analysis in R.” *Bioinformatics* 33.13, pp. 2063–2064. DOI: [10.1093/bioinformatics/btx099](https://doi.org/10.1093/bioinformatics/btx099).

Halachev, K., Bast, H., **Albrecht, F.**, Lengauer, T., and Bock, C. (2012). “EpiExplorer: live exploration and global analysis of large epigenomic datasets.” *Genome Biology* 13.10, R96. DOI: [10.1186/gb-2012-13-10-r96](https://doi.org/10.1186/gb-2012-13-10-r96).

List, M., Ebert, P., and **Albrecht, F.** (2017). “Ten simple rules for developing usable software in computational biology.” *PLoS Comput Biol* 13 (1), e1005265. DOI: [10.1371/journal.pcbi.1005265](https://doi.org/10.1371/journal.pcbi.1005265).

References

- Adams, D., Altucci, L., Antonarakis, S. E., Balles-teros, J., Beck, S., Bird, A., Bock, C., Boehm, B., Campo, E., Caricasole, A., *et al.* (2012). "BLUEPRINT to decode the epigenetic signature written in blood." *Nature biotechnology* 30.3, p. 224.
- Akhavan-Niaki, H. and Samadani, A. A. (2013). "DNA methylation and cancer development: molecular mechanism." *Cell biochemistry and biophysics* 67.2, pp. 501–513.
- Albrecht, F., List, M., Bock, C., and Lengauer, T. (2016). "DeepBlue epigenomic data server: programmatic data retrieval and analysis of epigenome region sets." *Nucleic Acids Research*. doi: 10.1093/nar/gkw211.
- Albrecht, F., List, M., Bock, C., and Lengauer, T. (2017). "DeepBlueR: large-scale epigenomic analysis in R." *Bioinformatics* 33.13, pp. 2063–2064.
- Anderson, S. (1981). "Shotgun DNA sequencing using cloned DNase I-generated fragments." *Nucleic acids research* 9.13, pp. 3015–3027.
- Arbel, H., Basu, S., Fisher, W. W., Hammonds, A. S., Wan, K. H., Park, S., Weiszmman, R., Booth, B. W., Keranen, S. V., Henriquez, C., *et al.* (2019). "Exploiting regulatory heterogeneity to systematically identify enhancers with high accuracy." *Proceedings of the National Academy of Sciences* 116.3, pp. 900–908.
- Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., *et al.* (2000). "Gene ontology: tool for the unification of biology." *Nature genetics* 25.1, p. 25.
- Assenov, Y., Müller, F., Lutsik, P., Walter, J., Lengauer, T., and Bock, C. (2014). "Comprehensive analysis of DNA methylation data with RnBeads." *Nature Methods* 11.11, pp. 1138–1140. doi: 10.1038/nmeth.3115.
- Bard, J., Rhee, S. Y., and Ashburner, M. (2005). "An ontology for cell types." *Genome biology* 6.2, R21.
- Barski, A., Cuddapah, S., Cui, K., Roh, T.-Y., Schones, D. E., Wang, Z., Wei, G., Chepelev, I., and Zhao, K. (2007). "High-resolution profiling of histone methylations in the human genome." *Cell* 129.4, pp. 823–837.
- Bernstein, B. E., Mikkelsen, T. S., Xie, X., Kamal, M., Huebert, D. J., Cuff, J., Fry, B., Meissner, A., Wernig, M., Plath, K., *et al.* (2006). "A bivalent chromatin structure marks key developmental genes in embryonic stem cells." *Cell* 125.2, pp. 315–326.
- Blankenberg, D., Kuster, G. V., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., Nekrutenko, A., and Taylor, J. (2010). "Galaxy: a web-based genome analysis tool for experimentalists." *Current protocols in molecular biology*, pp. 19–10.
- Bock, C., Halachev, K., Büch, J., and Lengauer, T. (2009). "EpiGRAPH: user-friendly software for statistical analysis and prediction of (epi)genomic data." *Genome Biology* 10.2, R14. doi: 10.1186/gb-2009-10-2-r14.
- Bock, C., Tomazou, E. M., Brinkman, A. B., Müller, F., Simmer, F., Gu, H., Jäger, N., Gnirke, A., Stunnenberg, H. G., and Meissner, A. (2010). "Quantitative comparison of genome-wide DNA methylation mapping technologies." *Nature Biotechnology* 28.10, pp. 1106–1114. doi: 10.1038/nbt.1681.
- Bock, C. (2012). "Analysing and interpreting DNA methylation data." *Nature Reviews Genetics* 13.10, pp. 705–719. doi: 10.1038/nrg3273.
- Boyer, L. A., Lee, T. I., Cole, M. F., Johnstone, S. E., Levine, S. S., Zucker, J. P., Guenther, M. G., Kumar, R. M., Murray, H. L., Jenner, R. G., *et al.* (2005). "Core transcriptional regulatory circuitry in human embryonic stem cells." *cell* 122.6, pp. 947–956.
- Breeze, C. E., Paul, D. S., Dongen, J. van, Butcher, L. M., Ambrose, J. C., Barrett, J. E., Lowe, R., Rakan, V. K., Iotchkova, V., Frontini, M., *et al.* (2016). "eFORGE: a tool for identifying cell type-specific signal in epigenomic data." *Cell reports* 17.8, pp. 2137–2150.
- Buenrostro, J. D., Wu, B., Litzenburger, U. M., Ruff, D., Gonzales, M. L., Snyder, M. P., Chang, H. Y., and Greenleaf, W. J. (2015). "Single-cell chromatin accessibility reveals principles of regulatory variation." *Nature* 523.7561, pp. 486–490. doi: 10.1038/nature14590.
- Bujold, D., Lima Morais, D. A. de, Gauthier, C., Côté, C., Caron, M., Kwan, T., Chen, K. C., Laperle, J., Markovits, A. N., Pastinen, T., *et al.* (2016). "The international human epigenome consortium data portal." *Cell systems* 3.5, pp. 496–499.
- Calo, E. and Wysocka, J. (2013). "Modification of enhancer chromatin: what, how, and why?" *Molecular cell* 49.5, pp. 825–837.
- Cervera, R., Ramos, A., Lluch, A., and Climent, J. (2016). "DNA Methylation in Breast Cancer." *Epigenetic Biomarkers and Diagnostics*. Elsevier, pp. 297–312.

- Cheng, J. B. and Russell, D. W. (2004). "Mammalian wax biosynthesis I. Identification of two fatty acyl-coenzyme A reductases with different substrate specificities and tissue distributions." *Journal of biological Chemistry* 279.36, pp. 37789–37797.
- Chen, X., Xu, H., Yuan, P., Fang, F., Huss, M., Vega, V. B., Wong, E., Orlov, Y. L., Zhang, W., Jiang, J., *et al.* (2008). "Integration of external signaling pathways with the core transcriptional network in embryonic stem cells." *Cell* 133.6, pp. 1106–1117.
- Chèneby, J., Gheorghe, M., Artufel, M., Mathelier, A., and Ballester, B. (2017). "ReMap 2018: an updated atlas of regulatory regions from an integrative analysis of DNA-binding ChIP-seq experiments." *Nucleic acids research* 46.D1, pp. D267–D275.
- Ciccarone, F., Tagliatesta, S., Caiafa, P., and Zampieri, M. (2018). "DNA methylation dynamics in aging: how far are we from understanding the mechanisms?" *Mechanisms of ageing and development* 174, pp. 3–17.
- Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., *et al.* (2009). "Biopython: freely available Python tools for computational molecular biology and bioinformatics." *Bioinformatics* 25.11, pp. 1422–1423.
- Colaprico, A., Silva, T. C., Olsen, C., Garofano, L., Cava, C., Garolini, D., Sabedot, T. S., Malta, T. M., Pagnotta, S. M., Castiglioni, I., *et al.* (2015). "TCGAbiolinks: an R/Bioconductor package for integrative analysis of TCGA data." *Nucleic acids research* 44.8, e71–e71.
- Consortium, G. O. (2018). "The Gene Ontology resource: 20 years and still GOing strong." *Nucleic Acids Research* 47.D1, pp. D330–D338.
- Creyghton, M. P., Cheng, A. W., Welstead, G. G., Kooistra, T., Carey, B. W., Steine, E. J., Hanna, J., Lodato, M. A., Frampton, G. M., Sharp, P. A., *et al.* (2010). "Histone H3K27ac separates active from poised enhancers and predicts developmental state." *Proceedings of the National Academy of Sciences* 107.50, pp. 21931–21936.
- Crujeiras, A. B. and Diaz-Lagares, A. (2016). "DNA methylation in obesity and associated diseases." *Epigenetic biomarkers and diagnostics*. Elsevier, pp. 313–329.
- Cusanovich, D. A., Daza, R., Adey, A., Pliner, H. A., Christiansen, L., Gunderson, K. L., Steemers, F. J., Trapnell, C., and Shendure, J. (2015). "Multiplex single cell profiling of chromatin accessibility by combinatorial cellular indexing." *Science (New York, NY)* 348.6237, pp. 910–914. doi: 10.1126/science.1257581.
- Dahl, C. and Guldberg, P. (2003). "DNA methylation analysis techniques." *Biogerontology* 4.4, pp. 233–250.
- Deaton, A. M. and Bird, A. (2011). "CpG islands and the regulation of transcription." *Genes & development* 25.10, pp. 1010–1022.
- Derrien, T., Johnson, R., Bussotti, G., Tanzer, A., Djebali, S., Tilgner, H., Guernec, G., Martin, D., Merkel, A., Knowles, D. G., Lagarde, J., Veeravalli, L., Ruan, X., Ruan, Y., Lassmann, T., Carninci, P., Brown, J. B., Lipovich, L., Gonzalez, J. M., Thomas, M., Davis, C. A., Shiekhattar, R., Gingeras, T. R., Hubbard, T. J., Notredame, C., Harrow, J., and Guigó, R. (2012). "The GENCODE v7 catalog of human long noncoding RNAs: analysis of their gene structure, evolution, and expression." *Genome Research* 22.9, pp. 1775–1789. doi: 10.1101/gr.132159.111.
- DeVries, A. and Vercelli, D. (2016). "DNA Methylation Biomarkers in Asthma and Allergy." *Epigenetic Biomarkers and Diagnostics*. Elsevier, pp. 331–350.
- Diaz-Lagares, A., Mendez-Gonzalez, J., Hervas, D., Saigi, M., Pajares, M. J., Garcia, D., Crujeiras, A. B., Pio, R., Montuenga, L. M., Zulueta, J., *et al.* (2016). "A novel epigenetic signature for early diagnosis in lung cancer." *Clinical Cancer Research*.
- Donlin, M. J. (2009). "Using the generic genome browser (GBrowse)." *Current Protocols in Bioinformatics* 28.1, pp. 9–9.
- Ebert, P., Müller, F., Nordström, K., Lengauer, T., and Schulz, M. H. (2015). "A general concept for consistent documentation of computational analyses." *Database* 2015. doi: 10.1093/database/bav050.
- ENCODE Project Consortium (2004). "The ENCODE (ENCyclopedia Of DNA Elements) Project." *Science (New York, NY)* 306.5696, pp. 636–640. doi: 10.1126/science.1105136.
- ENCODE Project Consortium (2012). "An integrated encyclopedia of DNA elements in the human genome." *Nature* 489.7414, pp. 57–74. doi: 10.1038/nature11247.
- Ernst, J., Kheradpour, P., Mikkelsen, T. S., Shores, N., Ward, L. D., Epstein, C. B., Zhang, X., Wang, L., Issner, R., Coyne, M., *et al.* (2011). "Mapping and analysis of chromatin state dynamics in nine human cell types." *Nature* 473.7345, p. 43.
- Ernst, J. and Kellis, M. (2012). "ChromHMM: automating chromatin-state discovery and characterization." *Nature methods* 9.3, p. 215.
- Ernst, J. and Kellis, M. (2017). "Chromatin-state discovery and genome annotation with ChromHMM." *Nature protocols* 12.12, p. 2478.

- Escalona, M., Rocha, S., and Posada, D. (2016). "A comparison of tools for the simulation of genomic next-generation sequencing data." *Nature Reviews Genetics* 17.8, p. 459.
- Farnel, S. and Shiri, A. (2014). "Metadata for research data: current practices and trends." *International Conference on Dublin Core and Metadata Applications*, pp. 74–82.
- Farlik, M., Halbritter, F., Müller, F., Choudry, F. A., Ebert, P., Klughammer, J., Farrow, S., Santoro, A., Ciaurro, V., Mathur, A., Uppal, R., Stunnenberg, H. G., Ouwehand, W. H., Laurenti, E., Lengauer, T., Frontini, M., and Bock, C. (2016). "DNA Methylation Dynamics of Human Hematopoietic Stem Cell Differentiation." *Cell Stem Cell* 19.6, pp. 808–822. doi: 10.1016/j.stem.2016.10.019.
- Farnham, P. J. (2009). "Insights from genomic profiling of transcription factors." *Nature Reviews Genetics* 10.9, p. 605.
- Favorov, A., Mularoni, L., Cope, L. M., Medvedeva, Y., Mironov, A. A., Makeev, V. J., and Wheelan, S. J. (2012). "Exploring massive, genome scale datasets with the GenometriCorr package." *PLoS computational biology* 8.5, e1002529.
- Fejes, A. P., Jones, M. J., and Kobor, M. S. (2014). "DaVIE: database for the visualization and integration of epigenetic data." *Frontiers in genetics* 5, p. 325.
- Fernández, X. M. and Birney, E. (2010). "Ensembl genome browser." *Vogel and Motulsky's Human Genetics*. Springer, pp. 923–939.
- Fernández, J. M., Torre, V. de la, Richardson, D., Royo, R., Puiggròs, M., Moncunill, V., Frangkogianni, S., Clarke, L., Flicek, P., Rico, D., et al. (2016). "The BLUEPRINT data analysis portal." *Cell systems* 3.5, pp. 491–495.
- Ferro, M., Ungaro, P., Cimmino, A., Lucarelli, G., Busetto, G. M., Cantiello, F., Damiano, R., and Terracciano, D. (2017). "Epigenetic signature: A new player as predictor of clinically significant prostate cancer (PCa) in patients on active surveillance (AS)." *International journal of molecular sciences* 18.6, p. 1146.
- Friedman, J. H., Hastie, T., and Tibshirani, R. (n.d.). "glmnet: lasso and elastic-net regularized generalized linear models, 2010b." URL <http://CRAN.R-project.org/package=glmnet>. R package version (), pp. 1–1.
- Friedman, J., Hastie, T., and Tibshirani, R. (2009). "glmnet: Lasso and elastic-net regularized generalized linear models." *R package version* 1.4.
- Friedman, J., Hastie, T., and Tibshirani, R. (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33.1, pp. 1–22. URL: <http://www.jstatsoft.org/v33/i01/>.
- Garber, M., Grabherr, M. G., Guttman, M., and Trapnell, C. (2011). "Computational methods for transcriptome annotation and quantification using RNA-seq." *Nature Methods* 8.6, pp. 469–477. doi: doi:10.1038/nmeth.1613.
- Garoufallou, E. and Papatheodorou, C. (2014). "A critical introduction to metadata for e-science and e-research." *International Journal of Metadata, Semantics and Ontologies* 9.1, pp. 1–4.
- García-Giménez, J. L., Ushijima, T., and Tollefsbol, T. O. (2016). "Epigenetic biomarkers: new findings, perspectives, and future directions in diagnostics." *Epigenetic biomarkers and diagnostics*. Elsevier, pp. 1–18.
- Gardiner-Garden, M. and Frommer, M. (1987). "CpG Islands in Vertebrate Genomes." *Journal of molecular biology* 196.2, pp. 261–282. doi: 10.1016/0022-2836(87)90689-9.
- García-Giménez, J. L. (2015). *Epigenetic biomarkers and diagnostics*. Academic Press.
- Gates, L. A., Shi, J., Rohira, A. D., Feng, Q., Zhu, B., Bedford, M. T., Sagum, C. A., Jung, S. Y., Qin, J., Tsai, M.-J., et al. (2017). "Acetylation on histone H3 lysine 9 mediates a switch from transcription initiation to elongation." *Journal of Biological Chemistry*, jbc-M117.
- Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., et al. (2004a). "Bioconductor: open software development for computational biology and bioinformatics." *Genome biology* 5.10, R80.
- Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A. J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J. Y. H., and Zhang, J. (2004b). "Bioconductor: open software development for computational biology and bioinformatics." *Genome Biology* 5.10, R80. doi: 10.1186/gb-2004-5-10-r80.
- Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., et al. (2005a). "Galaxy: a platform for interactive large-scale genome analysis." *Genome research* 15.10, pp. 1451–1455.
- Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W. J., and Nekrutenko, A. (2005b). "Galaxy: A platform for interactive large-scale genome analysis." *Genome Research* 15.10, pp. 1451–1455. doi: 10.1101/gr.4086505.

- Giresi, P. G., Kim, J., McDaniel, R. M., Iyer, V. R., and Lieb, J. D. (2007). "FAIRE (Formaldehyde-Assisted Isolation of Regulatory Elements) isolates active regulatory elements from human chromatin." *Genome research* 17.6, pp. 877–885.
- Giresi, P. G. and Lieb, J. D. (2009). "Isolation of active regulatory elements from eukaryotic chromatin using FAIRE (Formaldehyde Assisted Isolation of Regulatory Elements)." *Methods* 48.3, pp. 233–239.
- Goecks, J., Nekrutenko, A., and Taylor, J. (2010). "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences." *Genome biology* 11.8, R86.
- Goldberg, A. D., Allis, C. D., and Bernstein, E. (2007). "Epigenetics: A Landscape Takes Shape." *Cell* 128.4, pp. 635–638. doi: [10.1016/j.cell.2007.02.006](https://doi.org/10.1016/j.cell.2007.02.006).
- Goodwin, S., McPherson, J. D., and McCombie, W. R. (2016). "Coming of age: ten years of next-generation sequencing technologies." *Nature Reviews Genetics* 17.6, p. 333.
- Hahne, F., Durinck, S., Ivanek, R., Mueller, A., Lianoglou, S., Tan, G., and Parsons, L. (2012). *Gviz: Plotting data and annotation information along genomic coordinates*. R package version 1.13.2.
- Hahne, F. and Ivanek, R. (2016). "Visualizing genomic data using Gviz and bioconductor." *Statistical Genomics*. Springer, pp. 335–351.
- Halachev, K., Bast, H., Albrecht, F., Lengauer, T., and Bock, C. (2012). "EpiExplorer: live exploration and global analysis of large epigenetic datasets." *Genome Biology* 13.10, R96. doi: [10.1186/gb-2012-13-10-r96](https://doi.org/10.1186/gb-2012-13-10-r96).
- Harris, R. A. et al. (2010). "Comparison of sequencing-based methods to profile DNA methylation and identification of monoallelic epigenetic modifications." *Nature Biotechnology* 28.10, pp. 1097–1105. doi: [10.1038/nbt.1682](https://doi.org/10.1038/nbt.1682).
- Harrow, J. et al. (2012). "GENCODE: the reference human genome annotation for The ENCODE Project." *Genome Research* 22.9, pp. 1760–1774. doi: [10.1101/gr.135350.111](https://doi.org/10.1101/gr.135350.111).
- Heintzman, N. D., Stuart, R. K., Hon, G., Fu, Y., Ching, C. W., Hawkins, R. D., Barrera, L. O., Van Calcar, S., Qu, C., Ching, K. A., et al. (2007). "Distinct and predictive chromatin signatures of transcriptional promoters and enhancers in the human genome." *Nature genetics* 39.3, p. 311.
- Heinig, M., Colomé-Tatché, M., Taudt, A., Rintisch, C., Schafer, S., Pravenec, M., Hubner, N., Vingron, M., and Johannes, F. (2015). "histoneHMM: Differential analysis of histone modifications with broad genomic footprints." *BMC bioinformatics* 16.1, p. 60.
- Honsho, M. and Fujiki, Y. (2017). "Plasmalogen homeostasis—regulation of plasmalogen biosynthesis and its physiological consequence in mammals." *FEBS letters* 591.18, pp. 2720–2729.
- Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T., et al. (2002). "The Ensembl genome database project." *Nucleic acids research* 30.1, pp. 38–41.
- Ierusalimsky, R. (2006). *Programming in Lua, Lua*.
- Ikedo, Y. and Nishimura, T. (2015). "The role of DNA methylation in transposable element silencing and genomic imprinting." *Nuclear Functions in Plant Transcription, Signaling and Development*. Springer, pp. 13–29.
- Illingworth, R. S., Gruenewald-Schneider, U., Webb, S., Kerr, A. R., James, K. D., Turner, D. J., Smith, C., Harrison, D. J., Andrews, R., and Bird, A. P. (2010). "Orphan CpG islands identify numerous conserved promoters in the mammalian genome." *PLoS genetics* 6.9, e1001134.
- Jenuwein, T. and Allis, C. D. (2001). "Translating the Histone Code." *Science (New York, NY)* 293.5532, pp. 1074–1080. doi: [10.1126/science.1063127](https://doi.org/10.1126/science.1063127).
- Johnson, W. E., Li, C., and Rabinovic, A. (2007). "Adjusting batch effects in microarray expression data using empirical Bayes methods." *Biostatistics* 8.1, pp. 118–127.
- John, S., Sabo, P. J., Canfield, T. K., Lee, K., Vong, S., Weaver, M., Wang, H., Vierstra, J., Reynolds, A. P., Thurman, R. E., et al. (2013). "Genome-Scale Mapping of DNase I Hypersensitivity." *Current protocols in molecular biology*, pp. 21–27.
- Jones, B. A., Varambally, S., and Arend, R. C. (2018). "Histone methyltransferase EZH2: a therapeutic target for ovarian Cancer." *Molecular cancer therapeutics* 17.3, pp. 591–602.
- Karolchik, D., Hinrichs, A. S., Furey, T. S., Roskin, K. M., Sugnet, C. W., Haussler, D., and Kent, W. J. (2004). "The UCSC Table Browser data retrieval tool." *Nucleic acids research* 32.suppl_1, pp. D493–D496.
- Karlič, R., Chung, H.-R., Lasserre, J., Vlahoviček, K., and Vingron, M. (2010). "Histone modification levels are predictive for gene expression." *Proceedings of the National Academy of Sciences* 107.7, pp. 2926–2931.
- Karmodiya, K., Krebs, A. R., Oulad-Abdelghani, M., Kimura, H., and Tora, L. (2012). "H3K9 and H3K14 acetylation co-occur at many gene regulatory elements, while H3K14ac marks a subset of inactive inducible promoters in mouse embryonic stem cells." *BMC genomics* 13.1, p. 424.

- Kelly, T. K., Liu, Y., Lay, F. D., Liang, G., Berman, B. P., and Jones, P. A. (2012). "Genome-wide mapping of nucleosome positioning and DNA methylation within individual DNA molecules." *Genome Research* 22.12, pp. 2497–2506. doi: [10.1101/gr.143008.112](https://doi.org/10.1101/gr.143008.112).
- Kent, W. J., Sugnet, C. W., Furey, T. S., Roskin, K. M., Pringle, T. H., Zahler, A. M., and Haussler, D. (2002). "The Human Genome Browser at UCSC." *Genome Research* 12.6, pp. 996–1006. doi: [10.1101/gr.229102](https://doi.org/10.1101/gr.229102).
- Khavari, D. A., Sen, G. L., and Rinn, J. L. (2010). "DNA methylation and epigenetic control of cellular differentiation." *Cell Cycle* 9.19, pp. 3880–3883.
- Khare, S. P., Habib, F., Sharma, R., Gadewal, N., Gupta, S., and Galande, S. (2011). "Histone—a relational knowledgebase of human histone proteins and histone modifying enzymes." *Nucleic acids research* 40.D1, pp. D337–D342.
- Kirmizis, A., Bartley, S. M., Kuzmichev, A., Margueron, R., Reinberg, D., Green, R., and Farnham, P. J. (2004). "Silencing of human polycomb target genes is associated with methylation of histone H3 Lys 27." *Genes & development* 18.13, pp. 1592–1605.
- Knowles, D. G., Röder, M., Merkel, A., and Guigó, R. (2013). "Grape RNA-Seq analysis pipeline environment." *Bioinformatics* 29.5, pp. 614–621.
- Koch, C. M., Andrews, R. M., Flicek, P., Dillon, S. C., Karaöz, U., Clelland, G. K., Wilcox, S., Beare, D. M., Fowler, J. C., Couttet, P., et al. (2007). "The landscape of histone modifications across 1% of the human genome in five human cell lines." *Genome research* 17.6, pp. 691–707.
- Koon, H. B., Ippolito, G. C., Banham, A. H., and Tucker, P. W. (2007). "FOXP1: a potential therapeutic target in cancer." *Expert opinion on therapeutic targets* 11.7, pp. 955–965.
- Koohy, H., Down, T. A., Spivakov, M., and Hubbard, T. (2014). "A Comparison of Peak Callers Used for DNase-Seq Data." *PLoS ONE* 9.5, e96303. doi: [10.1371/journal.pone.0096303](https://doi.org/10.1371/journal.pone.0096303).
- Ku, C. S., Naidoo, N., Wu, M., and Soong, R. (2011). "Studying the epigenome using next generation sequencing." *Journal of medical genetics*, jmedgenet-2011.
- Kurdyukov, S. and Bullock, M. (2016). "DNA methylation analysis: choosing the right method." *Biology* 5.1, p. 3.
- Lander, E. S. et al. (2001). "Initial sequencing and analysis of the human genome." *Nature* 409.6822, pp. 860–921. doi: [10.1038/35057062](https://doi.org/10.1038/35057062).
- Lawrence, M., Huber, W., Pages, H., Aboyoun, P., Carlson, M., Gentleman, R., Morgan, M. T., and Carey, V. J. (2013). "Software for computing and annotating genomic ranges." *PLoS computational biology* 9.8, e1003118.
- Lay, F. D., Kelly, T. K., and Jones, P. A. (2018). "Nucleosome Occupancy and Methylome Sequencing (NOME-seq)." *DNA Methylation Protocols*. Springer, pp. 267–284.
- Layer, R. M., Pedersen, B. S., DiSera, T., Marth, G. T., Gertz, J., and Quinlan, A. R. (2018). "GIGGLE: a search engine for large-scale integrated genome analysis." *Nature methods* 15.2, p. 123.
- Leek, J. T. and Storey, J. D. (2007). "Capturing Heterogeneity in Gene Expression Studies by Surrogate Variable Analysis." *PLoS Genetics* 3.9, pp. 1724–1735. doi: [10.1371/journal.pgen.0030161](https://doi.org/10.1371/journal.pgen.0030161).
- Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., Langmead, B., Johnson, W. E., Geman, D., Baggerly, K., and Irizarry, R. A. (2010). "Tackling the widespread and critical impact of batch effects in high-throughput data." *Nature Reviews Genetics* 11.10, p. 733.
- Leek, J. T., Johnson, W. E., Parker, H. S., Jaffe, A. E., and Storey, J. D. (2012a). "The sva package for removing batch effects and other unwanted variation in high-throughput experiments." *Bioinformatics* 28.6, pp. 882–883.
- Leek, J. T., Johnson, W. E., Parker, H. S., Jaffe, A. E., and Storey, J. D. (2012b). "The sva package for removing batch effects and other unwanted variation in high-throughput experiments." *Bioinformatics (Oxford, England)* 28.6, pp. 882–883. doi: [10.1093/bioinformatics/bts034](https://doi.org/10.1093/bioinformatics/bts034).
- Li, Y. and Tollefsbol, T. O. (2011). "DNA methylation detection: bisulfite genomic sequencing analysis." *Epigenetics Protocols*. Springer, pp. 11–21.
- Li, D., Zhang, B., Xing, X., and Wang, T. (2015). "Combining MeDIP-seq and MRE-seq to investigate genome-wide CpG methylation." *Methods* 72, pp. 29–40.
- Li, E., Beard, C., and Jaenisch, R. (1993). "Role for DNA methylation in genomic imprinting." *Nature* 366.6453, p. 362.
- Li, H. (2011). "Tabix: fast retrieval of sequence features from generic TAB-delimited files." *Bioinformatics* 27.5, pp. 718–719.
- Lokk, K., Modhukur, V., Rajashekar, B., Märtens, K., Mägi, R., Kolde, R., Koltšina, M., Nilsson, T. K., Vilo, J., Salumets, A., et al. (2014). "DNA methylome profiling of human tissues identifies global and tissue-specific methylation patterns." *Genome biology* 15.4, p. 3248.

- Malone, J., Holloway, E., Adamusiak, T., Kapushesky, M., Zheng, J., Kolesnikov, N., Zhukova, A., Brazma, A., and Parkinson, H. (2010). "Modeling sample variables with an Experimental Factor Ontology." *Bioinformatics* 26.8, pp. 1112–1118.
- Martens, J. H. and Stunnenberg, H. G. (2013). *BLUEPRINT: mapping human blood cell epigenomes*.
- Maunakea, A. K., Nagarajan, R. P., Bilenky, M., Ballinger, T. J., D'Souza, C., Fouse, S. D., Johnson, B. E., Hong, C., Nielsen, C., Zhao, Y., et al. (2010). "Conserved role of intragenic DNA methylation in regulating alternative promoters." *Nature* 466.7303, p. 253.
- Meissner, A., Gnirke, A., Bell, G. W., Ramsahoye, B., Lander, E. S., and Jaenisch, R. (2005). "Reduced representation bisulfite sequencing for comparative high-resolution DNA methylation analysis." *Nucleic acids research* 33.18, pp. 5868–5877.
- Metzker, M. L. (2010). "Sequencing technologies—the next generation." *Nature reviews genetics* 11.1, p. 31.
- Müller, F. (2017). "Analyzing DNA Methylation Signatures of Cell Identity." PhD thesis. Saarland University. doi: [doi:10.17617/2.2474737](https://doi.org/10.17617/2.2474737).
- Mungall, C. J., Torniai, C., Gkoutos, G. V., Lewis, S. E., and Haendel, M. A. (2012). "Uberon, an integrative multi-species anatomy ontology." *Genome biology* 13.1, R5.
- Muto, A., Ikeda, S., Lopez-Burks, M. E., Kikuchi, Y., Calof, A. L., Lander, A. D., and Schilling, T. F. (2014). "Nipbl and mediator cooperatively regulate gene expression to control limb development." *PLoS genetics* 10.9, e1004671.
- Nakajima, T., Enomoto, S., and Ushijima, T. (2008). "DNA methylation: a marker for carcinogen exposure and cancer risk." *Environmental health and preventive medicine* 13.1, p. 8.
- National Human Genome Research Institute (2016). *DNA Sequencing Costs: Data*. <https://www.genome.gov/27541954/dna-sequencing-costs-data/>. Accessed: May 2016.
- Neph, S., Kuehn, M. S., Reynolds, A. P., Haugen, E., Thurman, R. E., Johnson, A. K., Rynes, E., Maurano, M. T., Vierstra, J., Thomas, S., et al. (2012). "BEDOPS: high-performance genomic feature operations." *Bioinformatics* 28.14, pp. 1919–1920.
- Oki, S., Ohta, T., Shioi, G., Hatanaka, H., Ogasawara, O., Okuda, Y., Kawaji, H., Nakaki, R., Sese, J., and Meno, C. (2018a). "ChIP-Atlas: a data-mining suite powered by full integration of public ChIP-seq data." *EMBO reports*, e46255.
- Oki, S., Ohta, T., Shioi, G., Hatanaka, H., Ogasawara, O., Okuda, Y., Kawaji, H., Nakaki, R., Sese, J., and Meno, C. (2018b). "Integrative analysis of transcription factor occupancy at enhancers and disease risk loci in noncoding genomic regions." *bioRxiv*, p. 262899.
- Ong, C.-T. and Corces, V. G. (2011). "Enhancer function: new insights into the regulation of tissue-specific gene expression." *Nature Reviews Genetics* 12.4, pp. 283–293. doi: [10.1038/nrg2957](https://doi.org/10.1038/nrg2957).
- Pageaud, Y., Plass, C., and Assenov, Y. (2018). "Enrichment analysis with EpiAnnotator." *Bioinformatics* 34.10, pp. 1781–1783.
- Pardal, R., Molofsky, A., He, S., and Morrison, S. (2005). "Stem cell self-renewal and cancer cell proliferation are regulated by common networks that balance the activation of proto-oncogenes and tumor suppressors." *Cold Spring Harbor symposia on quantitative biology*. Vol. 70. Cold Spring Harbor Laboratory Press, pp. 177–185.
- Parle-Mcdermott, A. and Harrison, A. (2011). "DNA methylation: a timeline of methods and applications." *Frontiers in genetics* 2, p. 74.
- Park, P. J. (2009). "ChIP-seq: advantages and challenges of a maturing technology." *Nature Reviews Genetics* 10, pp. 669–680. doi: [10.1038/nrg2641](https://doi.org/10.1038/nrg2641).
- Pekowska, A., Benoukraf, T., Ferrier, P., and Spicuglia, S. (2010). "A unique H3K4me2 profile marks tissue-specific gene regulation." *Genome research* 20.11, pp. 1493–1502.
- Pu, M., Ni, Z., Wang, M., Wang, X., Wood, J. G., Helfand, S. L., Yu, H., and Lee, S. S. (2015). "Trimethylation of Lys36 on H3 restricts gene expression change during aging and impacts life span." *Genes & development* 29.7, pp. 718–731.
- Quinlan, A. R. and Hall, I. M. (2010). "BEDTools: a flexible suite of utilities for comparing genomic features." *Bioinformatics* 26.6, pp. 841–842.
- Quinlan, A. R. (2014). "BEDTools: the Swiss-army tool for genome feature analysis." *Current protocols in bioinformatics* 47.1, pp. 11–12.
- R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: <http://www.R-project.org/>.
- Ramírez, F., Dündar, F., Diehl, S., Grüning, B. A., and Manke, T. (2014). "deepTools: a flexible platform for exploring deep-sequencing data." *Nucleic acids research* 42.W1, W187–W191.
- Ramalho-Carvalho, J., Henrique, R., and Jerónimo, C. (2016). "DNA Methylation Alterations as Biomarkers for Prostate Cancer." *Epigenetic Biomarkers and Diagnostics*. Elsevier, pp. 275–296.

- Ramírez, F., Ryan, D. P., Grüning, B., Bhardwaj, V., Kilpert, F., Richter, A. S., Heyne, S., Dündar, F., and Manke, T. (2016). "deepTools2: a next generation web server for deep-sequencing data analysis." *Nucleic acids research* 44.W1, W160–W165.
- Reddy, M. (2011). *API design for C++*. Morgan Kaufmann. ISBN: 9780123850034.
- Risso, D., Ngai, J., Speed, T. P., and Dudoit, S. (2014). "Normalization of RNA-seq data using factor analysis of control genes or samples." *Nature biotechnology* 32.9, p. 896.
- Roadmap Epigenomics Consortium *et al.* (2015). "Integrative analysis of 111 reference human epigenomes." *Nature* 518.7539, pp. 317–330. doi: 10.1038/nature14248.
- Roberts, A. and Pachter, L. (2013). "Streaming fragment assignment for real-time analysis of sequencing experiments." *Nature methods* 10.1, p. 71.
- Sandoval, J. and Serra, P. L. (2016). "DNA Methylation Biomarkers in Lung Cancer." *Epigenetic Biomarkers and Diagnostics*. Elsevier, pp. 259–273.
- Sarmiento, O. F., Digilio, L. C., Wang, Y., Perlin, J., Herr, J. C., Allis, C. D., and Coonrod, S. A. (2004). "Dynamic alterations of specific histone modifications during early murine development." *Journal of cell science* 117.19, pp. 4449–4459.
- Schatz, M. C. and Langmead, B. (2013). "The DNA data deluge: Fast, efficient genome sequencing machines are spewing out more data than geneticists can analyze." *Ieee Spectrum* 50.7, pp. 28–33.
- Sharp, A. J., Stathaki, E., Migliavacca, E., Brahmachary, M., Montgomery, S. B., Dupre, Y., and Antonarakis, S. E. (2011). "DNA methylation profiles of human active and inactive X chromosomes." *Genome research*.
- Sheffield, N. C. and Bock, C. (2016). "LOLA: enrichment analysis for genomic region sets and regulatory elements in R and Bioconductor." *Bioinformatics (Oxford, England)* 32.4, pp. 587–589. doi: 10.1093/bioinformatics/btv612.
- Shi, G. and Jin, Y. (2010). "Role of Oct4 in maintaining and regaining stem cell pluripotency." *Stem cell research & therapy* 1.5, p. 39.
- Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2011). "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent." *Journal of Statistical Software* 39.5, pp. 1–13. URL: <http://www.jstatsoft.org/v39/i05/>.
- Simon, J. M., Giresi, P. G., Davis, I. J., and Lieb, J. D. (2012). "Using formaldehyde-assisted isolation of regulatory elements (FAIRE) to isolate active regulatory DNA." *Nature protocols* 7.2, p. 256.
- Simon, J. M., Giresi, P. G., Davis, I. J., and Lieb, J. D. (2013). "A Detailed Protocol for Formaldehyde-Assisted Isolation of Regulatory Elements (FAIRE)." *Current protocols in molecular biology*, pp. 21–26.
- Singh, R., Lanchantin, J., Robins, G., and Qi, Y. (2016). "DeepChrome: deep-learning for predicting gene expression from histone modifications." *Bioinformatics* 32.17, pp. i639–i648.
- Skelly, A. C., Dettori, J. R., and Brodt, E. D. (2012). "Assessing bias: the importance of considering confounding." *Evidence-based spine-care journal* 3.01, pp. 9–12.
- Skinner, M. E., Uzilov, A. V., Stein, L. D., Mungall, C. J., and Holmes, I. H. (2009). "JBrowse: a next-generation genome browser." *Genome research*, gr-094607.
- Sloan, C. A., Chan, E. T., Davidson, J. M., Mal-ladi, V. S., Strattan, J. S., Hitz, B. C., Gabdank, I., Narayanan, A. K., Ho, M., Lee, B. T., *et al.* (2015). "ENCODE data at the ENCODE portal." *Nucleic acids research* 44.D1, pp. D726–D732.
- Song, L. and Crawford, G. E. (2010). "DNase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells." *Cold Spring Harbor Protocols* 2010.2, pdb-prot5384.
- Stalker, J., Gibbins, B., Meidl, P., Smith, J., Spooner, W., Hotz, H.-R., and Cox, A. V. (2004). "The Ensembl Web site: mechanics of a genome browser." *Genome research* 14.5, pp. 951–955.
- Stein, L. D., Mungall, C., Shu, S., Caudy, M., Mangone, M., Day, A., Nickerson, E., Stajich, J. E., Harris, T. W., Arva, A., *et al.* (2002). "The generic genome browser: a building block for a model organism system database." *Genome research* 12.10, pp. 1599–1610.
- Stevens, M., Cheng, J. B., Li, D., Xie, M., Hong, C., Maire, C. L., Ligon, K. L., Hirst, M., Marra, M. A., Costello, J. F., and Wang, T. (2013). "Estimating absolute methylation levels at single-CpG resolution from methylation enrichment and restriction enzyme sequencing methods." *Genome Research* 23.9, pp. 1541–1553. doi: 10.1101/gr.152231.112.
- Stunnenberg, H. G., International Human Epigenome Consortium, and Hirst, M. (2016). "The International Human Epigenome Consortium: A Blueprint for Scientific Collaboration and Discovery." *Cell* 167.5, pp. 1145–1149. doi: 10.1016/j.cell.2016.11.007.
- Tang, Q., Cheng, J., Cao, X., Surowy, H., and Burwinkel, B. (2016). "Blood-based DNA methylation as biomarker for breast cancer: a systematic review." *Clinical epigenetics* 8.1, p. 115.
- Telesnitsky, A. and Goff, S. (1997). "Reverse transcriptase and the generation of retroviral DNA."

- Teschendorff, A. E. and Relton, C. L. (2018). "Statistical and integrative system-level analysis of DNA methylation data." *Nature Reviews Genetics* 19.3, p. 129.
- Thomas, R., Thomas, S., Holloway, A. K., and Pollard, K. S. (2016). "Features that define the best ChIP-seq peak calling algorithms." *Briefings in bioinformatics* 18.3, pp. 441–450.
- Toska, E. and Sanz, F. J. C. (2016). "Genome-Wide Techniques for the Study of Clinical Epigenetic Biomarkers." *Epigenetic Biomarkers and Diagnostics*. Elsevier, pp. 119–135.
- Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., Van Baren, M. J., Salzberg, S. L., Wold, B. J., and Pachter, L. (2010). "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation." *Nature biotechnology* 28.5, p. 511.
- Tsompas, M. and Buck, M. J. (2014). "Chromatin accessibility: a window into the genome." *Epigenetics & chromatin* 7.1, p. 33.
- Venter, J. C. *et al.* (2001). "The Sequence of the Human Genome." *Science (New York, NY)* 291.5507, pp. 1304–1351. doi: [10.1126/science.1058040](https://doi.org/10.1126/science.1058040).
- Wang, J., Kong, L., Gao, G., and Luo, J. (2012). "A brief introduction to web-based genome browsers." *Briefings in Bioinformatics* 14.2, pp. 131–143.
- Weber, M., Davies, J. J., Wittig, D., Oakeley, E. J., Haase, M., Lam, W. L., and Schübeler, D. (2005). "Chromosome-wide and promoter-specific analyses identify sites of differential DNA methylation in normal and transformed human cells." *Nature genetics* 37.8, p. 853.
- Weber, M., Hellmann, I., Stadler, M. B., Ramos, L., Pääbo, S., Rebhan, M., and Schübeler, D. (2007). "Distribution, silencing potential and evolutionary impact of promoter DNA methylation in the human genome." *Nature genetics* 39.4, p. 457.
- Weintraub, H. and Groudine, M. (1976). "Chromosomal subunits in active genes have an altered conformation." *Science* 193.4256, pp. 848–856.
- Wen, Y., Cai, J., Hou, Y., Huang, Z., and Wang, Z. (2017). "Role of EZH2 in cancer stem cells: from biological insight to a therapeutic target." *Oncotarget* 8.23, p. 37974.
- Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York Inc. ISBN: 978-0-387-98140-6.
- Winer, D. (1999). *XML-RPC Specification*. URL: <http://xmlrpc.scripting.com/spec.html> (visited on 08/24/2018).
- Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., *et al.* (2013). "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud." *Nucleic acids research* 41.W1, W557–W561.
- Yen, A. and Kellis, M. (2015). "Systematic chromatin state comparison of epigenomes associated with diverse properties including sex and tissue type." *Nature communications* 6, p. 7973.
- Yong, W.-S., Hsu, F.-M., and Chen, P.-Y. (2016). "Profiling genome-wide DNA methylation." *Epigenetics & chromatin* 9.1, p. 26.
- Zerbino, D. R., Johnson, N., Juettemann, T., Wilder, S. P., and Flicek, P. (2013). "WiggleTools: parallel processing of large collections of genome-wide datasets for visualization and statistical analysis." *Bioinformatics* 30.7, pp. 1008–1009.
- Zerbino, D. R., Johnson, N., Juettemann, T., Wilder, S. P., and Flicek, P. (2014). "WiggleTools: parallel processing of large collections of genome-wide datasets for visualization and statistical analysis." *Bioinformatics (Oxford, England)* 30.7, pp. 1008–1009. doi: [10.1093/bioinformatics/btt737](https://doi.org/10.1093/bioinformatics/btt737).
- Zerbino, D. R., Wilder, S. P., Johnson, N., Juettemann, T., and Flicek, P. R. (2015). "The Ensembl Regulatory Build." *Genome Biology* 16.1, p. 56. doi: [10.1186/s13059-015-0621-5](https://doi.org/10.1186/s13059-015-0621-5).
- Zhang, Y., Liu, T., Meyer, C. A., Eeckhoute, J., Johnson, D. S., Bernstein, B. E., Nusbaum, C., Myers, R. M., Brown, M., Li, W., and Liu, X. S. (2008). "Model-based Analysis of ChIP-Seq (MACS)." *Genome Biology* 9.9, R137. doi: [10.1186/gb-2008-9-9-r137](https://doi.org/10.1186/gb-2008-9-9-r137).
- Zhang, B., Zhou, Y., Lin, N., Lowdon, R. F., Hong, C., Nagarajan, R. P., Cheng, J. B., Li, D., Stevens, M., Lee, H. J., *et al.* (2013). "Functional DNA methylation differences between tissues, cell types, and across individuals discovered using the M&M algorithm." *Genome research* 23.9, pp. 1522–1540.